OPTIMAL DESIGN OF PLA-BASED FSMs

A Thesis Submitted

In Partial Fulfilment of the Requirements

for the Degree of

MASTER OF TECHNOLOGY

by

S. RAMAN

to the

DEPARTMENT OF ELECTRICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY KANPUR

MAY 1992

Dedicated

to

my family members

&

shri. ADN

ACKNOWLEDGEMENTS

I wish to express my deep sense of gratitude to my thesis supervisor Prof.M.M.Hasan for his constant guidance and encouragement throughout the course of my thesis work.

I express my thanks to my colleagues R.Selvakumar, Ashish Singhai and to my seniors B.Prakash and V.S.Kakari for the interesting discussions we had on a broad range of topics.

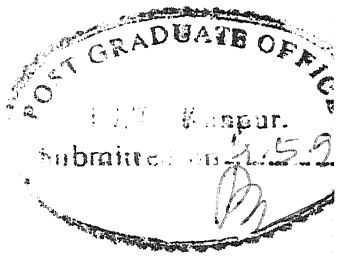
I acknowledge the warm and cheerful company of my micro-elec colleagues Subodh and Vandana and the inspiring suggestions of Sandeep Agarwal.

I express my heartfelt thanks to Ramesh Rao, Manoj Nair, Hariharan and all my other friends in bringing this thesis to a final shape. A special word of thanks to Dr.S.Narasimhan of Chemical Engg Dept and Bhoge, Natu and Abhay of Fibre Optics lab for permitting me to use their computing facilities during the course of the preparation of this thesis.

A special acknowledgement to MOTOROLA for their financial support during the final stages of my thesis work.

Many thanks are due to all my friends who made my stay at IIT-K a pleasurable one.

Last but not the least, a million thanks to my family members for making me what I am today.



C E R T I F I C A T E

It is certified that the work contained in the thesis entitled " **OPTIMAL DESIGN OF PLA-BASED FSMs** ", by **S.RAMAN,** has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

Dr. M.M.HASAN

Professor

Dept. of Electrical Engg.

I.I.T. KANPUR.

May 1992.

SYNOPSIS

This work addresses the problem of Optimal design of PLA-based Finite State Machines. State assignment profoundly affects the area,delay and testability of the combinational component of the FSM. Optimal design is aimed at minimizing the area of the PLA implementing the combinational component of the FSM. All previous approaches for PLA-based FSM synthesis use symbolic minimization and solve this problem as an encoding problem. These approaches use greater than minimum code length and hence results in higher area counts.

The present work deals with state assignment algorithms that use minimum code length for encoding the symbolic states of the FSM. These algorithms attempt to maximize the size and frequency of occurrence of the common cubes in the encoded machine prior to optimization. These algorithms encode the states based on the proximity relations in the Boolean space. Some methods to combine the various algorithms are also presented. Certain important optimality rules are specified. The delay in the PLA and the size of the test set are also determined. The effect of the algorithms on the area of the decomposed PLA are analyzed.

The algorithms were tested on a set of MCNC FSM benchmarks and the results are compared. The PLAs were tested for delay, test vector size and area after decomposition. The results show that the algorithms explained, compare favourably with symbolic minimization based approaches. Area count improvements were found to vary as much as 30% in some cases.

CONTENTS

LIST OF DIAGRAMS

LIST OF TABLES

Chapter 5

CHAPTER 1

INTRODUCTION

Very Large Scale Integration (VLSI) plays a major role in the development of complex electronic systems. As a result of the advances in semiconductor technology, an increasing number of devices can now be put on a single chip. The design methods for VLSI must hence be able to cope with the increased design complexity.

The design of a digital system can be viewed as a sequence of transformations of design representations at different levels of abstraction. The behavioural specifications of the system are described first by a functional description, using HDLs. This is transformed into a logic description, consisting of a net-list of logic gates, including storage elements. Subsequently logic synthesis is performed, which transforms and minimizes the representation, achieving a technology-independent set of logical expressions. The next step is to specify an electrical representation, according to an implementation technology, which is eventually transformed into a geometric layout of the integrated circuit implementing the given functionality.

The transformations involved in the synthesis of a VLSI chip depend on the design method used. Several "styles" are found today in the design arena. A fully custom design method is an adhoc implementation of a design optimized for a particular

application. In a gate-array design method, a circuit is implemented in silicon by personalizing a master array of uncommitted gates using a set of interconnections. The design is limited to routing the interconnections. A standard-cell design method requires partitioning the circuit into atomic units that are implemented by precommitted cells. Only placement and routing of the cells is to be done.

The design of VLSI circuits, using algorithmically generated macro-cells bridges the gap between custom and standard-cell design methods. Macro-cells can implement functional units that are specified by design parameters and by their functionality. They are usually highly regular and structured, allowing computer programs, called module generators, to produce the layout of a macro-cell from its functional description. Highly optimized and area-efficient modules can be designed in a short time. In particular Programmable Logic Array (PLA) macros have been shown to be very effective for designing both combinatorial and sequential functions.

PLAs allow for more efficient use of silicon area by representing a set of logic functions in a compressed yet regular form. A PLA implements two-stage combinational logic through an adjacent pair of rectangular arrays. Each position of the array is programmed by the presence or absence of a device. The functionality of a PLA can be represented by a 0-1 matrix, and the design and optimization of a PLA can be carried out directly in terms of this functional description.

PLA Implementation

A combinational logic function can be described by a logic cover. When designing a PLA implementation of a combinational function, the logic cover is represented by a pair of matrices, called input and output matrices.

The input and output matrices shown below have a corresponding PLA implementation shown in Fig.1.1.

```
--1--0   1000
-1-0--   0100
1----0   0001
1---1-   0100
0-----   0010
-----1   0001
```

Every input to the logic function (each column of the input matrix) corresponds to a pair of columns in the left part of the physical array. Every row of the input and output matrices, corresponds to a row of the physical array. Every input row corresponds to a logical product of some sets of inputs. Every output of the logic function (each column of the output matrix) corresponds to a column in the right part of the physical array. The implementation of a particular switching function is obtained by "programming" the PLA, i.e. by placing (or connecting) appropriate devices in the array in the input or column position specified by '1' or '0' . The input and output arrays are

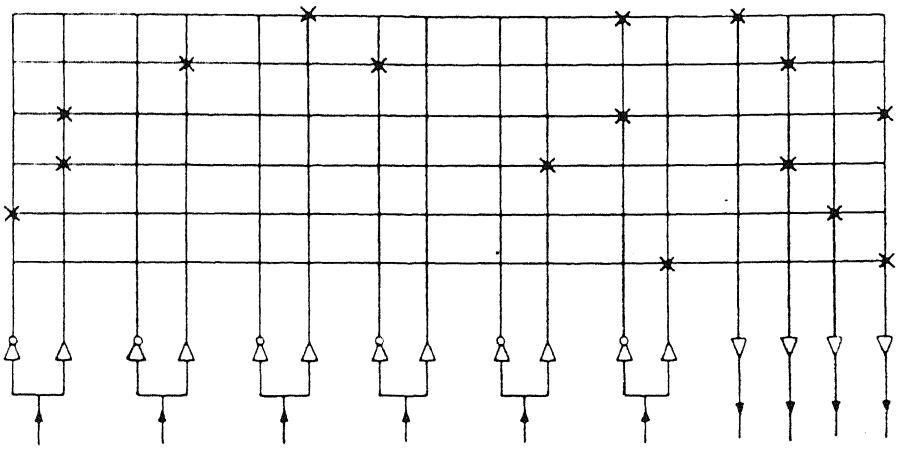


Fig.1.1 PLA physical implementation

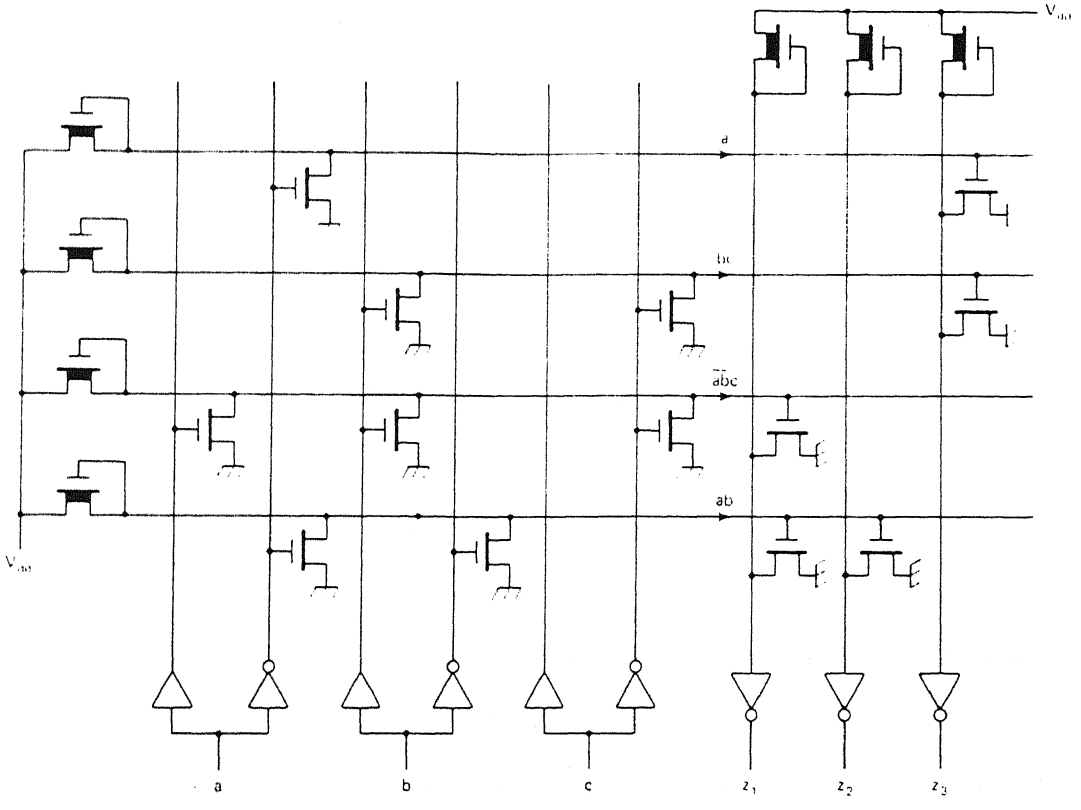


Fig.1.2 NOR-NOR NMOS PLA

referred to as the AND-plane and OR-plane respectively.

The key technological advantage of using a PLA in an integrated circuit technology relies on two things :-

i) the straight-forward mapping between the symbolic representation and its physical implementation, and,

ii) the compatibility with different technologies and modes of operation, such as AND-OR bipolar PLA, NOR-NOR NMOS PLA etc.

The NOR-NOR implementation is most common in VLSI MOS circuits since in MOS technology it is convenient to exploit the use of NOR gates. PLAs are implemented in the form of "sums-of-sums"(or more exactly complemented-sums-of-complemented-sums). The transformation from a sum-of-products representation can be easily carried out, with the additional requirement of output inverters. An example of such a NOR-NOR NMOS PLA implementing the following functions is shown in Fig.1.2.

$$z_1 = a\ b + !a\ !b\ c\ ;$$
$$z_2 = a\ b\ ;$$
$$z_3 = a + !b\ c\ ;$$

The structure of a CMOS PLA is very similar to the NMOS PLA. An example of a precharged CMOS PLA to realize the same functions z_1,z_2 and z_3 as given above is shown in Fig.1.3. When cl = 0 , the outputs z_1,$z_2$and z_3, as well as the outputs from the NAND stages, are precharged to 0 . When the clock comes high, some of

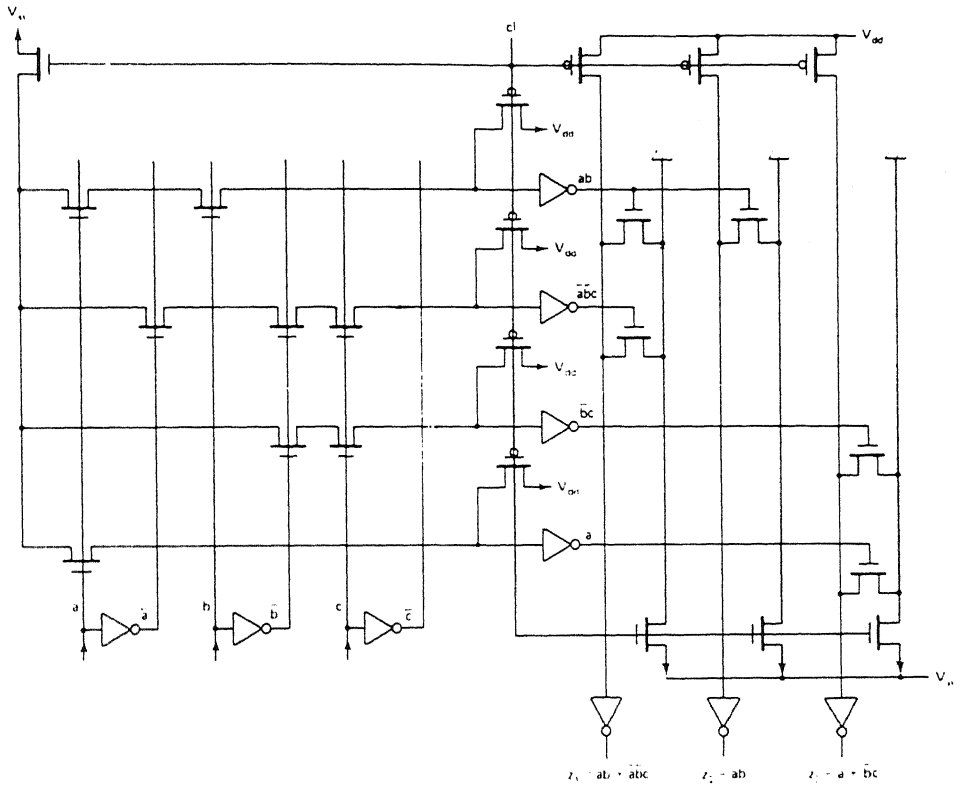


Fig.1.3 Precharged CMOS PLA

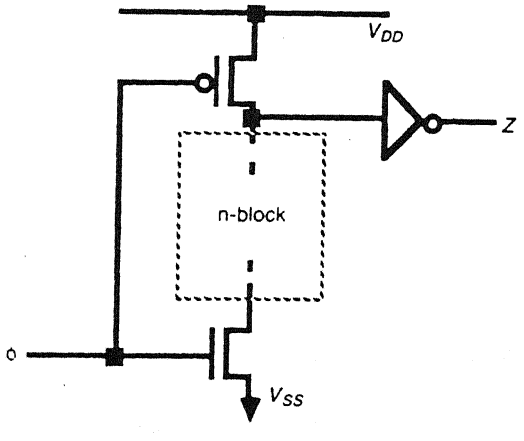


Fig.1.4 CMOS Domino logic

the output may change to 0 , depending on the logic implemented. Many a times, CMOS domino logic is used which when compared with the conventional CMOS logic, have smaller areas and lesser parasitic capacitances. The general CMOS domino logic structure is shown in Fig.1.4. Now attempts are being made towards the realization of BiCMOS PLAs, which combine the advantages of Bipolar and CMOS technologies.

Sequential logic functions can be represented as Finite State Machines (FSMs) and implemented by a combinational and a storage component. A regular array (such as PLA) can implement effectively the FSM combinational component and the storage elements by a series of flip-flops (such as D FFs). PLA-based FSMs are very popular in the control circuitry of large chips and is the focus of the present work. The efficient and optimal (in terms of area and delay) synthesis of PLA-based FSMs is a key area of active research for many decades.

1.2 Thesis Outline

The thesis is organized as follows.

Chapter 2 gives a detailed description of the definitions and notations used. A detailed introduction to PLA-based FSMs is given. A thorough review of the previous approaches in PLA-based FSM synthesis is made. The algebraic techniques proposed in the early past are discussed. Graph based techniques for state assignment and three important symbolic minimization based approaches are explained. The chapter ends with an explanation

for the need for a better state assignment procedure catered to PLA-based FSMs.

Chapter 3 gives an overview of the algorithms for state assignment. The basic factors which influence the encoding process and eventually the logic minimization step are explained. The state assignment algorithms are explained in the context of PLA-based FSMs and is illustrated with an example.

Chapter 4 deals with the various aspects of PLA synthesis and design. It begins with an analysis of the drawbacks and inadequacies of the algorithms explained in chapter 3. Then, some of the techniques used to minimize the inadequacies are explained. The various constraints which affect the optimal design of PLAs are discussed. The technique used is illustrated with an example. Then the chapter deals with delays and delay calculations in PLAs. A brief explanation on the testing of PLAs and the tool used to test the PLA is given. The chapter ends with a discussion on the need for decomposition and a description of the decomposition tool used.

Chapter 5 consists of results and discussions on it. The chapter begins with an introduction to the logic minimization tool used for analysis. The effect of state assignment on the area, delay, size of test sets needed and the area of the decomposed PLA are tabulated.

CHAPTER 2

PLA BASED FSMs

PLAs are attractive building blocks of a structured design methodology. Many industrial VLSI circuits, such as the Intel 8086, the Motorola 68000, the HP 32-bit and the Bell Mac-32 Microprocessors, use PLAs as building blocks. In particular, PLA-based FSMs are quite popular and are often used in the control circuitry of such large chips. These PLA-based FSMs can be designed very efficiently, because the properties of two-level combinational functions are well understood. In this chapter, a brief outline of the definitions, notations used, an introduction to PLA-based FSMs and a thorough review of previous techniques for synthesis of PLA-based FSMs are given.

2.1 Definitions and Notations

A variable is a symbol representing a single co-ordinate of the Boolean space (e.g., α). A literal is a variable or its negation(e.g., α or $!\alpha$). A cube is a set C of literals such that x is an element of C implies that !x is not an element of C (e.g. {a,b,!c} is a cube, and {a,!a} is not a cube). A cube represents the conjunction of its literals. The trivial cubes, writen as '0' and '1' represent the Boolean functions 0 and 1, respectively. An expression is a set f of cubes. For example, { {a},{b,!c} } is an expression consisting of the two cubes {a} and {b,!c}. An expression represents the disjunction of its cubes.

A cube may also be written as a bit vector on a set of variables with each bit position representing a distinct variable. The values taken by each bit can be 1,0, or 2 (don't care), signifying the true form, negated form, and nonexistence, respectively, of the variable corresponding to that position. A minterm is a cube with only 0 and 1 entries. Cubes can be classified based on the number of 2 entries in the cube. A cube with k entries or bits which take the value 2 is called a k-cube. A minterm thus is a 0-cube. A cube c_1 is said to cover (contain) another cube c_2, if c_1 evaluates to 1 for every minterm for which c_2 evaluates to 1. A supercube of a set of cubes, c_i is defined as the smallest cube containing all the minterms contained in c_i. The on-set of f is the set of minterms for which f evaluates to 1, and the don't care set (DC-set) is the set of minterms for which the value of the function is unspecified. An implicant of f is a cube that does not contain any minterm in the off-set of f. A prime-implicant of f is an implicant which is not contained by any other implicant of f.

In general, a logic function may have symbolic (also known as multiple-valued) input or output variables in addition to binary variables. Like binary variables, a symbolic variable also represents a single co-ordinate, with the difference that a symbolic variable can take a subset of values from a set, say P_i , that has a cardinality greater than two. When an output variable is symbolic, it implies that the variable can take a value from a set of values, when the input is some minterm. A function in which some variables are symbolic is known as a symbolic cover. A convenient way for representing a symbolic variable that can take

values from a set of cardinality n is to use an n-bit vector to depict a literal of that variable such that each position in the vector corresponds to a specific element of the set. A 1 in a position in the vector signifies the presence of an element in the literal while a 0 signifies the absence. This method of representation is commonly known as one-hot.

A Finite State Machine is an abstract model describing the synchronous sequential circuit. It is represented schematically as shown in Fig.2.1.

Formally, a FSM can be defined as a 5-tuple (X,Y,Z,δ λ) where

$X = \{ x_1, x_2, \ldots\ldots x_{|X|} \}$ is the set of primary input symbols,

$Y = \{ y_1, y_2, \ldots\ldots y_{|Y|} \}$ is the set of internal states,

$Z = \{ z_1, z_2, \ldots\ldots z_{|Z|} \}$ is the set of primary output symbols,

$\delta : X \times Y \rightarrow Y$ is the next state function, and,

$\lambda : X \times Y \rightarrow Z$ ($\lambda : Y \rightarrow Z$) is the output function for a Mealy (Moore) machine.

A Finite State Machine is represented by a State Transition Table/Graph (STT/STG) . A FSM is a STG $G(V,E,W(E))$ where V is the set of vertices corresponding to the set of states S, where N_S is the cardinality of S, an edge (v_i, v_j) joins v_i to v_j if there is a primary input that causes the FSM to evolve from state v_i to state v_j and $W(E)$ is a set of labels attached to each edge, each label carrying the information of the value of the input that

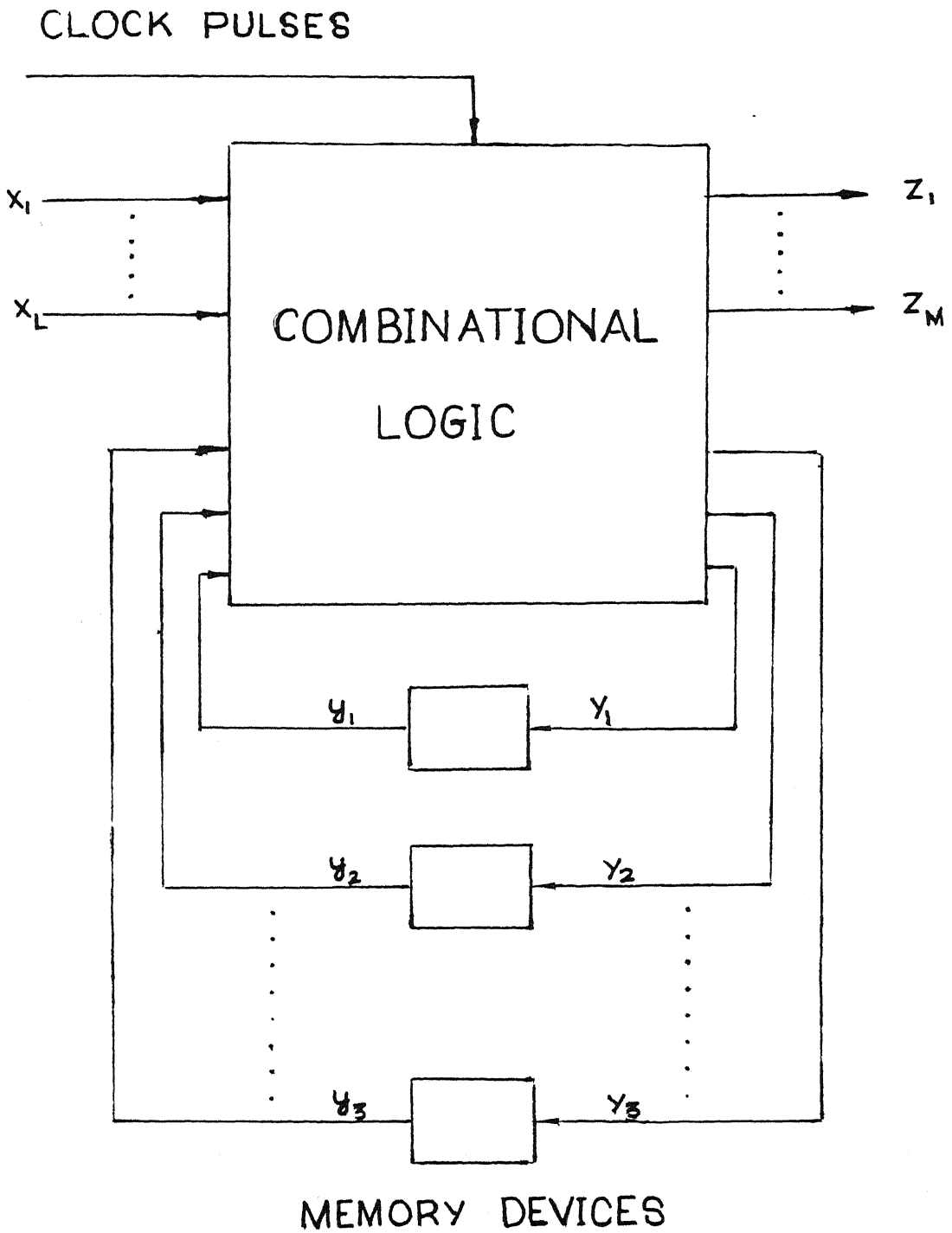


Fig.2.1 Schematic representation of a FSM.

caused the transition and the values of the primary outputs corresponding to that transition.

Alternatively, a FSM is a STT $T\langle I,S,O\rangle$ where I is the set of inputs, S is the set of states and O is the set of outputs. The primary inputs and outputs are in Boolean form. A row of the table corresponds to an edge in the transition graph. The table has as many rows as edges of the graph and as many columns as N_i+N_o+2 , where the '2' refers to the symbolic present and next states.

A finite state machine representation is said to be incompletely specified if the next state and/or output function are not specified for some input and/or present state. Else, the machine is said to be completely specified.

A starting or initial state is asumed to exist for a machine, also called the reset state. Given a logic-level machine with N_b flip flops, 2^{N_b} possible states exist in the machine. A state which can be reached from the reset state via some input vector sequence is called a valid state in the transition graph. The input sequence is called the justification sequence for that state. A state for which no justification sequence exists is called an invalid state.

A differentiating sequence for a pair of states q_1, q_2 of a machine M is a sequence of input vectors such that the last vector produces different outputs, when the sequence is applied to M, when M is initially in q_1 or when M is initially in q_2. Two states in a completely specified machine M are equivalent, if they do not possess a differentiating sequence. Two states in an

incompletely specified machine M are compatible, if they do not possess a differentiating sequence.

The Hamming distance between two codes is the number of bit positions at which the two codes differ from each other. Obviously, the Hamming distance can vary between 0 and the length of either code.

2.2 Structure of PLA-based FSMs

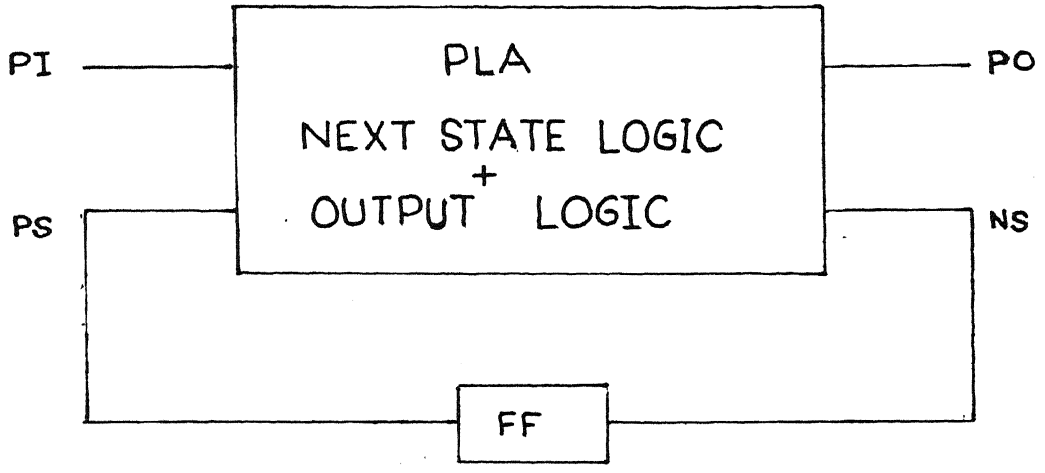
The PLA implementation of a FSM combinational component can satisfy two major requirements :-

i) regular and structured design that can be supported by CAD tools, and,

ii) size and performance of the silicon implementation.

Since PLAs are regular and structured, the entire FSM is regular allowing easy automation of design. A model of synchronous FSM using delay latches is shown in Fig.2.2.

Several techniques like logic minimization and topological compaction allow the design of area-effective PLA implementations. Therefore, PLA-based FSM design can be optimized with regard to silicon area requirement and subsequently to switching-time performance. The memory component of a FSM consists of a set of latches that store the machine state representation. The most commonly used type of latch is the D-latch. The operation is synchronized with a system clock to keep race-free design even when the circuit size is large.



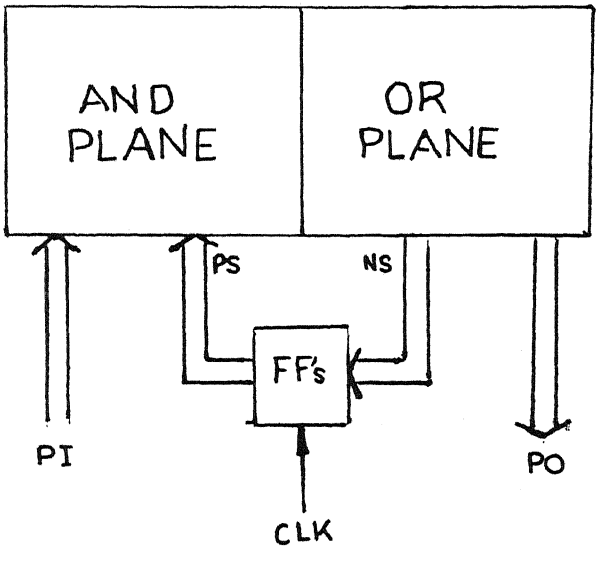


Fig.2.2 Synchronous FSM model using delay latches.

2.3 Synthesis of PLA-based FSMs

The automatic synthesis of a sequential circuit as a PLA-based FSM involves functional design, logic design, topological design and physical design. The step of logic design maps the functional description into a logic representation in terms of logic variables. A representation of the symbolic states in terms of Boolean variables, called state assignment is chosen. The complexity of the combinational component of the FSM depends heavily on the state assignment and selection of memory elements. PLA optimization aims at minimizing the area ocupied by the PLA and the delay through it (proportional to the number of product terms, to a first-order approximation) . The PLA area is proportional to the product of the number of rows times the number of columns. The optimum state assignment (or encoding) problem looks for the assignment corresponding to a PLA implementation of minimum area. The (minimum) number of rows is the cardinality of the (minimum) cover of the FSM combinational component according to a given assignment. The number of bits used to represent the states is related to the number of PLA columns. Therefore, the PLA area depends in a complex way on the state assignment.

While a first look at a PLA might suggest that area/delay is the main optimality constraint to be optmized, it is also true that testability, reliability and technology constraints are as equally important as the area of the PLA. A PLA is easily testable if the number of test vectors needed to test it is minimal. It has been shown that state assignment affects not only the eventual layout area and performance of a logic-level

sequential machine, but also has a profound effect on the testability of the machine as well [1]. Generating tests for sequential circuits is much harder than for combinational circuits, because of the inaccessability of the next-state and present-state lines. Hence complete/incomplete scan designs were adopted, with a huge penalty in the area. However, a far more efficient and robust approach is one that ensures a fully testable machine during synthesis, by constraining the state asignment process. The synthesized machine can also be made easily testable so that test sequences for the faults in the machine can be obtained more simply than using general sequential test generation. The area penalty incurred due to the constraints on the optimization/synthesis are relatively smaller than the scan designs.

2.4 Review of previous approaches

State Assignment is one of the oldest problems in automata theory. The state assignment problem is to find binary codes for symbolic states in a FSM so that the combinational logic that implements the FSM is optimum for some objective, usually logic area but sometimes delay and testability as well.

State Assignment is computationally complex. It has been shown that[2] state assignment is NP-hard. Hence a lot of heuristics have been suggested in the past. Because FSMs are usually given to logic optimizers after state asignment, one good source for state assignment heuristics is the relationship between state assignment and logic optimization. Hence there is a rich early literature on the state assignment problem. Some of the

prominent ones among them are reviewed below.

Hartmanis,Stearns, Karp and Kohavi [3,4,5] developed algebraic methods based on partition theory. Their approach was based on a reduced dependence criterion, which led to a good assignment. However, no theoretical result was presented that related reduced dependencies to optimal FSM implementations. Moreover, no systematic procedure was developed that could be used to encode large machines.

Armstrong[6] was the first to formulate the encoding problem as a graph-embedding problem, where a graph defines adjacency relations (in terms of Hamming distance) between the codes of the states to be preserved by a subgraph isomorphism on the encoding cube, with the objective to minimize the number of gates of the final implementation. His approach was ineffective, because i) he did not take into account the techniques of fast heuristic logic minimizers (heuristic logic minimization was not known at that time) , ii) he transformed the state assignment problem into a graph embedding problem, that only partially represented the state encoding problem, and iii) his graph embedding approach was ineffective.

Dolotta and McCluskey [7] intorudced the concept of codeable columns, used for finding near-optimal solutions. Their algorithm was able to estimate the complexity of the combinational component of a FSM, as a function of partial-length state assignments. However, their method was computationally efficient for small machines only. Their method was later improved by Weiner and Smith[8], Torng[8] and Story[8]. Curtis[8] considered the problem

of using different types of storage elements in relation to the state assignment problem. Tracey[8] and Saucier[9] addressed the state-assignment problem in connection with race-free asynchronous machine design. In Saucier's approach, the problem was reduced to a graph embedding problem. Each input defined a partition of the states (or of a subset of the states) by the successor relation. The states were assigned to vertices in the cube so that for each partition the images of all states in the same block formed a path (using, if necessary,states not also included in the partition or unused vertices, each for at most one block) disjoint from the ones associated to the other blocks. In terms of implementations of minimum area, these approaches suffered from a weak connection with the logic optimization steps after the encoding.

With the advent of improved techniques in multiple-valued logic minimization, the state-assignment problem took a new turn. Here, the states of the machine are represented as the set of possible values for a single multiple-valued variable. Logic minimization is applied on a symbolic representation of the combinational component of the machine. The efect of multiple-valued logic minimization is to group together the states that are mapped by some input into the same next state and assert the same output. A new combinatorial optimization problem arises (called FACE HYPERCUBE EMBEDDING) of assigning each of these sets(called input constraints) to subcubes of a Boolean k-cube, for a minimum k,in a way that each subcube contains all and only all the codes of the states included in the corresponding constraint. Symbolic minimization is a technique that yields a

minimal encoding-independent sum-of-products representation of a symbolic function. It builds up a directed acyclic graph, where the nodes are the next states and an edge (u,v) corresponds to the covering constraint (called the output constraint) that the code u covers bit-wise the code of v. The translation of the cover obtained by symbolic minimization into a compatible Boolean representation defines simultaneously a face hypercube embedding problem and an output covering problem (called ORDERED FACE HYPERCUBE EMBEDDING). The three most important programs using symbolic minimization for FSM synthesis are discussed below.

2.4.1 Keep Internal States Simple [8]

This program, one of the earliest, using symbolic minimization attempts to find an assignment of minimum code length among the assignments that minimize the number of rows of the PLA. This technique attempts to group together the state encodings in binary-valued logical implicants in the same way states as grouped in the minimal symbolic (multiple-valued-input) cover. In particular, it gives an encoding such that each symbolic implicant can be coded by one binary-valued implicant. For this asignment, it has been shown here that there exists a binary-valued cover of the FSM combinational component having as many implicants as the minimal symbolic cover. It gives an assignment such that the face-hypercube embedding requirements are satisfied. For this a constrained encoding problem is considered and has been proved that any solution to the constrained encoding problem is a state assignment such that the coded Boolean cover has the same cardinality as the minimal symbolic cover. Here the state assignment is restricted to one-to-one mappings between the state

set and a subset of the vertices of the Boolean hypercube, i.e., each state encoding is a 0-dimensional subspace. The encoding algorithm presented constructs a state code matrix which satisfies the constraints of the encoding as defined by the minimized symbolic cover and the final Boolean cover cardinality is no greater than the symbolic cover cardinality.

2.4.2 NOVA [10]

NOVA has a collection of algorithms for optimal state assignment of FSMs based on the solution of face hypercube embedding and ordered face hypercube embedding. It also uses an effective version of symbolic minimization that focusses on the output encoding problem also. iexact-code is an exact algorithm that finds an encoding satisfying all input constraints and minimizing the encoding length. The face hypercube embedding problem is transformed into subposet dimension problem. The algorithm finds an answer to subposet dimension by answering exactly subposet equivalence for increasing dimensions of the hypercube. ihybrid-code and igreedy-code are some heuristics for the computationally complex iexact-code. These heuristics maximize input constraint satisfaction for a given encoding length. iohybrid-code solves ordered face hypercube embedding. It's an adaptation of ihybrid-code that simultaneously maximizes input and output constraint satisfaction. These algorithms are shown to be much better than KISS[sec 2.4.1].

2.4.3 DIET [11]

DIET (Dichotomy-based symbolic Input Encoding Technique) gives a new theoretical formulation of the input encoding problem

based on the concept of compatibility of dichotomies. The input encoding problem is shown to be equivalent to a two-level logic minimization and three techniques to solve the encoding problem are discussed. These techniques are based on

i) techniques borrowed from classical logic minimization (generation of prime dichotomies and solving the covering problem) ,

ii) graph coloring applied to the graph of incompatibility of dichotomies, and

iii) extraction of essential prime dichotomies followed by graph coloring .

The input encoding problem is formulated in the same way as is done in KISS . However, it is solved with a more efficient method based on the concept of compatibility of dichotomies. A close analogy between logic minimization and input encoding problem is explained and used advantageously. DIET compares favourably with KISS and NOVA in terms of the encoding length and CPU time, while relatively occupying lesser memory.

2.4.4 Constrained synthesis procedures

Quite recently, testability has achieved equal significance in the synthesis arena[12]. In [13], a synthesis procedure which guaranteed fully testable irredundant combinational logic circuits was proposed. In [1], a procedure which produced a fully and easily testable logic-level sequential machine from a STG description was proposed. An optimal synthesis procedure that guarantees full nonscan testability under the stuck-at fault

model, with no associated area or performance penalty has been proposed in [14] . Due to the inherent structure of a PLA, stuck-at-fault models were found to be inadequate and an extended model, called the crosspoint-fault model was introduced. A procedure for constrained state assignment and logic optimization which guarantees testability for all combinationally irredundant crosspoint faults in a PLA-based FSM was introduced in [12] . In this procedure, each pair of states, which do not produce mutually nondominating primary outputs for atleast one primary input vector, are assigned mutually nondominating codes. Recently, Prakash[15], modified this procedure to cover a wide range of FSMs and presented improved results.

2.5 Need for a better Assignment

As has been discussed earlier, the area of the PLA depends both on the cardinality of the rows of the PLA and on the cardinality of the columns. The symbolic minimization based encoding techniques discussed above tries to assign binary codes to the symbolic states with a code length which is not necessarily the minimum. In fact, in most of the cases, it was found that the encoding length was greater than the minimum required, which is $[\log_2 n]$, where [x] is an integer nearest and greater or equal to x and n is the state set cardinality. The techniques using symbolic minimization discussed above thus produces at many times, a PLA whose area is not necessarily the minimum. This was because the number of columns in the PLA increases with increase in code length. However, these programs are efficient enough in limiting the number of rows of the PLA.

Hence there is a need for a state assignment procedure which will not only reduce the number of product terms, but also achieve this in a minimum code length and optionally implementing a PLA with a smaller delay as well. This means that a synthesis procedure that guarantees minimal product terms with a minimum code length will give a PLA satisfying these requirements. In the context of symbolic minimization based encoding technique, this may mean that satisfaction of all the input constraints is not fully accomplishable. In the next chapter, a synthesis procedure tuned towards achieving this goal is explained.

CHAPTER 3

STATE ASSIGNMENT

3.1 Introduction

The State Assignment problem is to find binary codes for symbolic states in a finite state machine so that the combinational logic that implements the machine is optimum for logic area and/or delay and testability as well. Many methods such as those that were discussed in the previous chapter use symbolic minimization and they attempt to solve the state assignment problem as a type of input encoding problem. These approaches used greater than minimum code length. In this chapter, algorithms for achieving an assignment aiming at a minimal area implementation of the combinational component of the FSM are given. Though these algorithms[17] were originally meant for multi-level logic, these are used as the basis for the state assignment of PLA-based FSMs.

The first step in FSM synthesis is state minimization. Quite often, the state transition description contains redundant states, i.e., states whose functions can be accomplished by other states. Since the number of memory elements required for a realization of the machine is directly related to the number of states, the minimization of the number of states (or the elimination of redundant states) in many cases reduce the complexity and cost of the realization. Also, the testability becomes easier if the machine does not contain redundant states. Hence state minimization is an important step. The state minimization procedure implemented in [16] is used.

In this chapter, the basic approach followed to obtain a good state assignment is first described. Then, a detailed account of the algorithms, illustrated by an example, is given.

3.2 Influence of Encoding on the cover of the PLA

A good state assignment heuristic is one that accurately models the logic optimization step that succeeds state assignment. Two-level logic functions are sums of cubes, so cubes are the most complex common factors that can be found in two-level functions. Multi-level logic can contain common factors that are themselves sums of cubes. Hence a heuristic that affects the size and number of common cubes is aptly suited for two-level implementations(PLAs).

The algorithms explained below are based on maximizing the number and size of common cubes that exist in the Boolean equations that describe the combinational logic part of the FSM after the states have been encoded but before logic optmization. The state assignment algorithms find pairs or clusters of states which if kept minimally distant in the Boolean space representing the encoding, results in a large number of common cubes in the Boolean network. There are two basic processes behind the influence of state assignment on the number of common cubes in the encoded state transition table (a two-level representation). They are discussed below.

Let the number of bits used for encoding be denoted by N_b. Let N_d denote the distance between two codes of length N_b.

Obviously N_d can vary between 0 and N_b .

Consider a state transition table fragment shown below.

1-	s0	s2	1
1-	s1	s2	1
11	s0	s1	0

In this STT two different present states s0 and s1 go to the same next state s2 for some common input combination. If the states s0 and s1 are assigned codes of distance N_d , then the lines of the next state s2 will have a common cube with $N_b - N_d$ literals.

Now consider another fragment in which a same present state s0 goes to two different next states s1 and s2. Then if the states s1 and s2 are assigned codes of distance N_d, then the present state s0 becomes a common cube for $(N_b - N_d)$ next state lines, whatsoever be the code of s0 .

Similar such relationships exist between other sets of states in the STT.

The input and output spaces (the first and the fourth fields of STT) also have an influence on the number of common cubes after encoding. If two different input combinations, i_1 and i_2, produce the same next state from different or same present states, then there is a common cube corresponding to $i_1 \cap i_2$ in the input space. Similarly, outputs asserted by different present states have common cubes corresponding to their intersections.

Thus, there is a large set of relationships between state encoding and the number/size of common cubes in the network prior to logic optimization. The gains, that can be obtained by coding a given pair of states with close codes so that single/multiple occurrences of common cubes can be extracted, can be estimated. Given these gains for each pair of states, an algorithm to find an encoding which maximizes the overall gain is implemented.

However, there is a slight complication in gain estimation. Firstly, while the number of literals in the common cubes can be found exactly, the number of occurrences of these cubes depends on the encoding of the next states. This problem may be reduced by using an average case analysis.

Thus the goal of the algorithms explained below is to find an encoding that maximizes the size/number of common cubes in the two-level description.

3.3 The Overall Strategy

The basic approach is to build a complete graph $G_M(V, E_M, W(E_M))$ where V is in one-to-one correspondence with the states of the finite state machine. E_M is a complete set of edges, i.e., every node is connected to every other node, and $W(E_M)$ represents the gains that can be achieved by coding the states joined by the corresponding arc as close as possible. These gains are statically and independently computed by

enumerating the direct relationships betwen the input, state and output spaces.

Then, the states are encoded, using this graph to provide the cost of an assignment of a state to a vertex of the Boolean hypercube.

A critical part of the approach is the generation of WCE_M. There are two algorithms discussed below for the generation of WCE_M. The first one called the fanout algorithm assigns the weights to the edges by taking into consideration the second and fourth fields of the STT. This algorithm attempts to maximize the size of the most frequently occuring common cubes in the encoded machine prior to optimization. The second algorithm assigns weights to the edges by taking into account the first and third fields of the STT and is called the fanin algorithm. This algorithm attempts to maximize the number of occurrences of the largest common cubes in the encoded machine prior to optimization. These two algorithms are based on the two different processes behind the influence of state assignment on the number of common cubes in the network, as described earlier in the previous section.

3.4 The Fanout Algorithm

This algorithm works on the output and the fanout of each state. Present states which assert similar outputs and produce similar sets of next states are given high edge weights (and

eventually close codes) so as to maximize the size of common cubes in the output and next state lines.

Description

(1) Construct a complete graph $G_H(V, E_H, W(E_H))$, with edge weight set empty. For each output, all the labels, $W(E_H)$ in the STG G, are scanned to identify the nodes which assert that output. N_o sets of weighted nodes which assert each output are constructed. If a node asserts the same output more than once, it has a correspondingly larger weight in the set.

(2) For each next state, sets of present states producing that next state are found (N_s sets are constructed).

The pseudocode below illustrates these steps of the algorithm. nw stores the weight of the nodes in each of the different sets.

```
for(i = 1; i ≤ N_o; i = i + 1) {
   foreach( edges e(v_k, v_l) ∈ G ) {
      if( W(e).output[i] is 1 ) {
         OUTPUT_SET_i = OUTPUT_SET_i ∪ v_k
         nw (OUTPUT_SET_i , v_k ) = (OUTPUT_SET_i , v_k ) + 1
      }
   }
}
```

```
foreach ( edges e(v_k, v_l) ∈ G ) {
    N_STATE_SET_l = N_STATE_SET_i ∪ v_k
    nw ( N_STATE_SET_l , v_k ) = nw (N_STATE_SET_l , v_k ) + 1
}
```

(3) Using these N_o OUTPUT_SET N_s N_STATE_SET sets of nodes, WCE_M) is constructed. The edge weight *we* is equal to the multiplication of the weights of the two nodes corresponding to it across all sets. The weights corresponding to the next states have a multiplicative factor equal to half the number of encoding bits $N_b/2$. The pseudocode for the calculation of *we* is shown below.

```
foreach( (v_k, v_l) ∈ G_M ) {
    for(i = 1; i ≤ N_s ;i = i + 1 )
        we(e_M(v_k, v_l)) = we(e_M(v_k, v_l)) + ( nw(N_STATE_SET_i, v_k)
                                * nw(N_STATE_SET_i, v_l) )
        we(e_M(v_k, v_l)) = we(e_M(v_k, v_l)) * N_b / 2
    for(i = 1 ;i ≤ N_o ;i = i + 1 )
        we(e_M(v_k, v_l)) = we(e_M(v_k, v_l)) + nw(OUTPUT_SET_i , v_k)
                                * nw( OUTPUT_SET_i, v_l)
}
```

The first step of the algorithm entails enumerating the relationships between the present states and the output space. If two different present states assert an output, it is possible to extract a common cube corresponding to the intersection of the two state codes. By constructing the N_o different output sets and counting the number of times a pair of states occur together in each output set, the algorithm effectively computes the number of occurrences of the common cube $X \cap Y$, for all states X and Y . Multiple assertions of same output for many input combinations by the same state is taken care of by the weight $nw()$. For two states that assert the same output a multiple number of times, each pair of edges will have the common cube. Accordingly, the weights $nw()$ are multiplied.

In the second step, the next states produced by each pair of present states are compared. A state pair which produces the same next state has an associated common cube corresponding to the pairwise intersection. The number of occurrences of this common cube is dependent on the number of 1's in the code of the next state. since the number of 1's in a state's code varies between 0 and N_b, an average of $N_b/2$ is taken as the multiplicative factor.

Given the number of occurrences of different common cubes in the machine, this algorithm assigns weights so as to maximize the size of the most frequently occurring cubes.

3.5 The Fanin Algorithm

The fanin algorithm operates on the input and fanin for each state. Next states which are produced by similar inputs and similar sets of present states are given high edge weights (and eventually close codes) so as to maximize the number of common cubes in the next state lines.

Description

(1) Construct a complete graph $G_M(V, E_M, W(E_M))$ with the edge weight set empty. N_s sets of weighted next states which fanout from each present state in G are constructed as shown below. nw stores the weight of each node in all the sets. The pseudocode is given below.

```
foreach( edge e(v_k, v_l) ∈ G ) {
    P_STATE_SET_k = P_STATE_SET_k ∪ v_l
    nw(P_STATE_SET_k, v_l) = nw(P_STATE_SET_k, v_l) + 1
}
```

(2) For each input, sets of next states are identified which are produced when the input is 1 and when the input is 0 . $2 * N_i$ such sets are constructed as shown below.

```
for( i=1 ;i≤ N_i ;i =i + 1) {
    foreach(edge e(v_k,v_l) ∈ G) {
        if (w(e).input[i] is 1 ) {
            INPUT_SET_i^ON = INPUT_SET_i^ON ∪ v_l
            new(INPUT_SET_i^ON,v_l) = (INPUT_SET_i^ON,v_l ) + 1
        }
        if ( w(e).input[i] is 0 ) {
            INPUT_SET_i^OFF = INPUT_SET_i^OFF ∪ v_l
            new(INPUT_SET_i^OFF,v_l) = (INPUT_SET_i^OFF,v_l ) + 1
        }
    }
}
```

(3) The weights on the edges in the graph, we, are found using the N_i INPUT_SETs of ON and OFF, N_s P_STATE_SET sets of nodes as illustrated below. Between each pair of nodes in G_M, an edge with weight equal to the multiplication of the weights of the two nodes across all the present state sets (scaled by N_b) and all the input sets is added.

The first step of the algorithm entails enumerating the relationships between the input and next state space. A next state produced by two different input combinations has a common cube in the intersection of the two input combinations. The size of this cube can be found. By counting the 2 * N_i different

input sets and counting the number of times a pair of states occurs together in each input set, the algorithm computes similarity relationships between all next state pairs in terms of the inputs. Giving next state pairs that are produced by similar inputs high edge weights will result in maximizing the number of occurrences of the largest common input cubes in the next state lines.

In the second step, the present states producing each pair of next states are compared. If two different next states are produced by the same present state, the state is common to some next state lines. The number of occurrences of this common cube is dependent on the intersection of the two next state codes. To maximize the number of occurrences of these cubes, next state pairs which have many common present states are given correspondingly high edge weights. Since each of these cubes have N_b literals, there is a multiplying factor of N_b while combining the weights computed in the two steps.

Given the sizes of the different common cubes in the machine, this algorithm assigns weights so as to maximize the number of occurrences of these cubes.

3.6 The Embedding Algorithm

The fanin and fanout algorithms presented in the previous sections generate a complete weighted graph to guide the state encoding process. The next step is to assign the actual codes to

states according to the analysis performed by the fanin and fanout algorithms. This problem is a classical combinatorial optimization problem called graph embedding problem. Here G_M has to be embedded in the Boolean hypercube so that the adjacency relations identified by G_M are satisfied in an optimal way. This problem is NP-complete and an heuristic called the *wedge clustering* algorithm is explained below.

The wedge clustering heuristic algorithm tries to minimize the cost function

$$\sum_{i=1}^{N_s} \sum_{l=i+1}^{N_s} we(e_M(v_i, v_l)) * \text{dist}\,(\,enc(v_i), enc(v_l)\,)$$

while assigning codes to the nodes.

Here the v_k are the vertices, *we* is the edge weight, *e*, between the vertices and *enc* is the encoding length of the vertex. The function dist() returns the distance between two binary codes.

The graphs generated by the fanout and fanin algorithms have a certain structure associated with them especially for large machines. In these graphs, typically small groups of states exist that are strongly connected internally (edges between states in the same group have high edge weights) but weakly connected externally (edges between states not in the same cluster have low weights). The wedge clustering heuristic algorithm exploits the nature of the graph by attempting to identify strongly connected clusters and assigning states within each cluster with uni-distant codes.

The embedding algorithm proceeds as follows. Clusters of nodes with the cardinality of the cluster no greater than N_b+1 and consisting of edges of maximum total weight are identified in the graph. Given the graph, the identification of these clusters is as follows - a node, $v_1 \in G_M$, with the maximum sum of weights of any N_b connected edges is identified. These nodes may be labeled as $y_1, y_2, \ldots y_{N_b}$ which correspond to the N_b edges from v_1 and v_1 are assigned minimally distant codes from the unassigned codes(v_1 may have been assigned already, so may the other y_i). A maximum of N_b nodes are chosen so that y_i can be (possibly) assigned unidistant codes from v_1 and all the edges connected to it are deleted from the graph. The node selection/code assignment process is repeated till all the nodes are assigned codes. The pseudocode below illustrates the procedure.

```
GG = G_M

while (GG is not empty)
{

        select v_1 ∈ GG , y_i ∈ GG   so  Σ_{i=1}^{N_b} wec( e_M(v_1, v_i) ) is
    maximum

    assign the  y_i  and  v_1  minimally distant codes from
    unassigned codes

    GG = GG - v_1

}
```

It has been proved in [17] that this heursitic is the optimal one for this particular cost function. It is also shown in [17] that the worst case time complexity of this heuristic is $O(N_s^2 (\log\langle N_s \rangle + \langle N_b \rangle))$ and is quite fast.

3.7 An Example

The fanin and fanout algorithms for weighted-graph construction and the embedding algorithm for hypercube embedding are illustrated, by an example, below.

The statistics of the example are :-

Name	:	ex6
Primary Inputs	:	5
Primary Outputs	:	8
States	:	8

The STT is shown in Fig.3.1

Firstly, a complete graph is constructed with vertices corresponding to the states and edge weights initially equal to zero. Since there are 8 states, a minimum of 3 bits are needed to encode the states.

Fanin algorithm :

This algorithm calculates the weights of the edges in the graph by forming two seperate classes of sets called the input sets and present state sets.

```
#The name of this benchmark is "ex6" .
.i 5
.o 8
.p 34
.s 8
11--- s1 s3 10111000
00--- s1 s2 11000000
10--- s1 s4 00101000
0-0-- s2 s2 11000000
--1-- s2 s5 00001110
110-- s2 s3 10111000
100-- s2 s4 00101000
10--- s3 s4 00111000
00--- s3 s2 11010000
11--- s3 s3 10111000
01--- s3 s6 00110101
010-- s4 s6 00100101
--1-- s4 s7 00101000
110-- s4 s3 10111000
000-- s4 s2 11000000
100-- s4 s4 00101000
1-10- s5 s8 10000100
--0-- s5 s2 11000000
--11- s5 s8 10000100
0-10- s5 s5 00001110
----1 s6 s2 11000001
10--0 s6 s4 00101001
00--0 s6 s2 11000001
11--0 s6 s3 10111001
01--0 s6 s6 00100101
--0-- s7 s2 11000000
101-- s7 s7 00101000
011-- s7 s6 00100101
111-- s7 s3 10111000
001-- s7 s2 11000000
101-- s8 s7 00101000
--0-- s8 s2 11000000
0-1-- s8 s8 10000100
111-- s8 s3 10111000
.e
```

Fig.3.1 STT of MCNC FSM benchmark ex6.

For each input, there is a false-input set and the true-input set, corresponding to whether the input is 0 or 1 respectively. Thus there are $2 * N_i$ input sets. For each present state, there is a present state set. The sets are constructed as follows. For each input set, sets of next states are identified which are produced when the input is of corresponding polarity. Consider $i_2(0)$. The set is given by

$$i_2(0) = \{ s1^0, s2^5, s3^0, s4^5, s5^0, s6^0, s7^2, s8^0 \}$$

All the input sets are constructed similarly.

For each present-state, sets of next states which are produced are identified. For example,

$$s2 = \{ s1^0, s2^1, s3^1, s4^1, s5^1, s6^0, s7^0, s8^0 \}$$

This implies that in the STG description, there is a transition from s2 to each of s2,s3,s4 and s5 once, as indicated by the weights. All the N_s sets of present state sets are constructed similarly. All the input sets and the present state sets of this example are shown in Fig.3.2.

The edge weights are calculated as follows. Between each pair of nodes in the constructed graph, an edge with weight equal to the multiplication of the weights of the two nodes across all the present state sets (scaled by the N_b) and all the input sets is added. Consider for example, the weight calculation for the edge between s2 and s4 ; the edge weight is given by

INPUT SETS :-

	s1	s2	s3	s4	s5	s6	s7	s8
i1(Ø)	Ø	6	Ø	Ø	1	4	Ø	1
i1(1)	Ø	Ø	7	5	Ø	Ø	2	1
i2(Ø)	Ø	5	Ø	5	Ø	Ø	2	Ø
i2(1)	Ø	Ø	7	Ø	Ø	4	Ø	Ø
i3(Ø)	Ø	5	2	2	Ø	1	Ø	Ø
i3(1)	Ø	1	2	Ø	2	1	3	3
i4(Ø)	Ø	Ø	Ø	Ø	1	Ø	Ø	1
i4(1)	Ø	Ø	Ø	Ø	Ø	Ø	Ø	1
i5(Ø)	Ø	1	1	1	Ø	1	Ø	Ø
i5(1)	Ø	1	Ø	Ø	Ø	Ø	Ø	Ø

PRESENT_STATE SETS :-

	s1	s2	s3	s4	s5	s6	s7	s8
s1	Ø	1	1	1	Ø	Ø	Ø	Ø
s2	Ø	1	1	1	1	Ø	Ø	Ø
s3	Ø	1	1	1	Ø	1	Ø	Ø
s4	Ø	1	1	1	Ø	1	1	Ø
s5	Ø	1	Ø	Ø	1	Ø	Ø	2
s6	Ø	2	1	1	Ø	1	Ø	Ø
s7	Ø	2	1	Ø	Ø	1	1	Ø
s8	Ø	1	1	Ø	Ø	Ø	1	1

Fig.3.2 Input and present state sets of ex6.

$$e_{24} = (6*0 + 0*7 + 5*5 + 0*0 + 5*2 + 1*0 + 0*0 + 0*0 + 1*1 + 1*0) + 3(1*1 + 1*1 + 1*1 + 1*1 + 1*0\ 2*1 + 2*0 + 1*0)$$

$$= 54$$

The other edge weights are calculated similarly. The complete weighted graph is shown in Fig.3.4a.

Fanout algorithm

This algorithm works on the output and fanout of each state. The output sets and next state sets are constructed as follows.

For each output, sets of present states asserting that output are found out. For example,

$$O_5 = \{ s1^2, s2^3, s3^2, s4^3, s5^1, s6^2, s7^2, s8^2 \}$$

All the 8 output sets are constructed similarly.

For each next state, sets of present states producing that next state are found. Say, for s8 as the next state, s5 goes to s8 twice and s8 stays in itself once in the STT. Hence,

$$s8 = \{ s1^0, s2^0, s3^0, s4^0, s5^2, s6^0, s7^0, s8^1 \}$$

The output and next states are shown in Fig.3.3. The edge weight is equal to the multiplication of the weights of the two nodes across all output sets and next state sets (scaled by half the no of bits). The weight of the edge between s1 and s2 is

OUTPUT_SETS :-

	s1	s2	s3	s4	s5	s6	s7	s8
o1	2	2	2	2	3	3	3	3
o2	1	1	1	1	1	2	2	1
o3	2	2	3	4	Ø	3	3	2
o4	1	1	4	1	Ø	1	1	1
o5	2	3	2	3	1	2	2	2
o6	Ø	1	1	1	3	1	1	1
o7	Ø	1	Ø	Ø	1	Ø	Ø	Ø
o8	Ø	Ø	1	1	Ø	5	1	Ø

NEXT_STATE SETS :-

	s1	s2	s3	s4	s5	s6	s7	s8
s1	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
s2	1	1	1	1	1	2	2	1
s3	1	1	1	1	Ø	1	1	1
s4	1	1	1	1	Ø	1	Ø	Ø
s5	Ø	1	Ø	Ø	1	Ø	Ø	Ø
s6	Ø	Ø	1	1	Ø	1	1	Ø
s7	Ø	Ø	Ø	1	Ø	Ø	1	1
s8	Ø	Ø	Ø	Ø	2	Ø	Ø	1

Fig.3.3 Output and next state sets of ex6.

s1 :	s2	0.0	s3	0.0	s4	0.0	s5	0.0	s6	0.0	s7	0.0	s8	0.0	
s2 :	s3	40.0	s4	54.0	s5	14.0	s6	49.0	s7	25.0	s8	18.0			
s3 :	s4	55.0	s5	7.0	s6	45.0	s7	29.0	s8	16.0					
s4 :	s5	3.0	s6	12.0	s7	23.0	s8	5.0							
s5 :	s6	6.0	s7	6.0	s8	14.0									
s6 :	s7	9.0	s8	7.0											
s7 :	s8	14.0													

Fig.3.4a Weighted graph of ex6 produced by fanin algorithm.

s1	s2	20.5	s3	23.5	s4	24.5	s5	10.5	s6	25.0	s7	23.5	s8	19.0	
s2	s3	26.5	s4	28.5	s5	17.0	s6	28.0	s7	26.5	s8	22.0			
s3	s4	35.0	s5	13.5	s6	38.5	s7	33.0	s8	25.0					
s4	s5	14.5	s6	40.5	s7	36.5	s8	27.5							
s5	s6	19.0	s7	19.0	s8	19.5									
s6	s7	42.0	s8	27.5											
s7	s8	29.0													

Fig.3.4b Weighted graph of ex6 produced by fanout algorithm.

States	Assignment
s2	011
s1	110
s5	111
s8	101
s7	001
s4	010
s6	000
s3	100
DCARE	---

Fig.3.4c Final Assignment by wedge clustering algorithm.

$$e_{12} = (2*2 + 1*1 + 2*2 + 1*1 + 2*3 + 0*1 + 0*1 + 0*0) +$$

$$(3/2)(0*0 + 1*1 + 1*1 + 1*1 + 0*1 + 0*0\ 0*0\ 0*0)$$

$$= 20.5$$

The other edge weights are calculated similarly and the complete weighted graph is shown in Fig.3.4b.

Embedding algorithm

Given a complete weighted graph, this algorithm works as follows. A cluster is formed with head node as the node with maximum set of any N_b edge weights and the tail nodes as the nodes connected to the head node via the maximum weight edges. This requires sorting across all the nodes and sorting all the nodes. This cluster is given to the routine *give_codes()*, which tries to give minimal Hamming distance code to all the members of the cluster. If the head node and all the tail nodes are not yet assigned codes, then a code (randomly) is given to the head node. Then all the unassigned tail nodes are given minimal distance code from the head node.

For this example, the complete weighted graph is shown in Fig.3.4b. After sorting, the graph is shown in Fig.3.4a. The first cluster is

s6 : s7(42.0) s4(40.5) s3(38.5)

The head is s6 and 3 edges of maximum weight, 121.0, is chosen. The 3 nodes connected to s6 via these 3 maximum weight edges are s7,s4 and s3 with respective weights given in brackets.

```
s6    : s7  42.0    s4  40.5    s3  38.5
s6 : max_wt = 121.0 s7(42.0) s4(40.5) s3(38.5) s2(28.0) s8(27.5) s1(25.0) s5(19.0) s6(-1.0)
s4 : max_wt = 112.0 s6(40.5) s7(36.5) s3(35.0) s2(28.5) s8(27.5) s1(24.5) s5(14.5) s4(-1.0)
s7 : max_wt = 111.5 s6(42.0) s4(36.5) s3(33.0) s8(29.0) s2(26.5) s1(23.5) s5(19.0) s7(-1.0)
s3 : max_wt = 106.5 s6(38.5) s4(35.0) s7(33.0) s2(26.5) s8(25.0) s1(23.5) s5(13.5) s3(-1.0)
s8 : max_wt = 84.0 s7(29.0) s6(27.5) s4(27.5) s3(25.0) s2(22.0) s5(19.5) s1(19.0) s8(-1.0)
s2 : max_wt = 83.0 s4(28.5) s6(28.0) s3(26.5) s7(26.5) s8(22.0) s1(20.5) s5(17.0) s2(-1.0)
s1 : max_wt = 73.0 s6(25.0) s4(24.5) s7(23.5) s3(23.5) s2(20.5) s8(19.0) s5(10.5) s1(-1.0)
s5 : max_wt = 57.5 s8(19.5) s7(19.0) s6(19.0) s2(17.0) s4(14.5) s3(13.5) s1(10.5) s5(-1.0)
```

```
s4    : s7  36.5    s3  35.0    s2  28.5
s4 : max_wt = 100.0 s7(36.5) s3(35.0) s2(28.5) s8(27.5) s1(24.5) s5(14.5) s6(40.5) s4(-1.0)
s7 : max_wt = 98.5 s4(36.5) s3(33.0) s8(29.0) s2(26.5) s1(23.5) s5(19.0) s6(42.0) s7(-1.0)
s3 : max_wt = 94.5 s4(35.0) s7(33.0) s2(26.5) s8(25.0) s1(23.5) s5(13.5) s6(38.5) s3(-1.0)
s2 : max_wt = 81.5 s4(28.5) s7(26.5) s3(26.5) s8(22.0) s1(20.5) s5(17.0) s6(28.0) s2(-1.0)
s8 : max_wt = 81.5 s7(29.0) s4(27.5) s3(25.0) s2(22.0) s5(19.5) s1(19.0) s6(27.5) s8(-1.0)
s1 : max_wt = 71.5 s4(24.5) s3(23.5) s7(23.5) s2(20.5) s8(19.0) s5(10.5) s6(25.0) s1(-1.0)
s5 : max_wt = 55.5 s8(19.5) s7(19.0) s2(17.0) s4(14.5) s3(13.5) s1(10.5) s6(19.0) s5(-1.0)
s6 : max_wt = 121.0 s7(42.0) s4(40.5) s3(38.5) s2(28.0) s8(27.5) s1(25.0) s5(19.0) s6(-1.0)
```

Fig.3.5 Weighted graphs explaining the flow of embedding

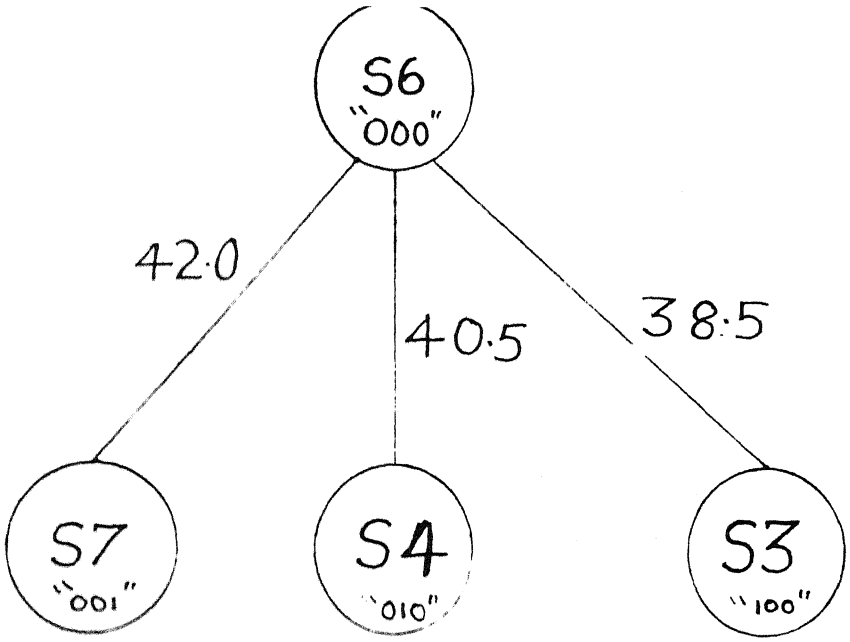
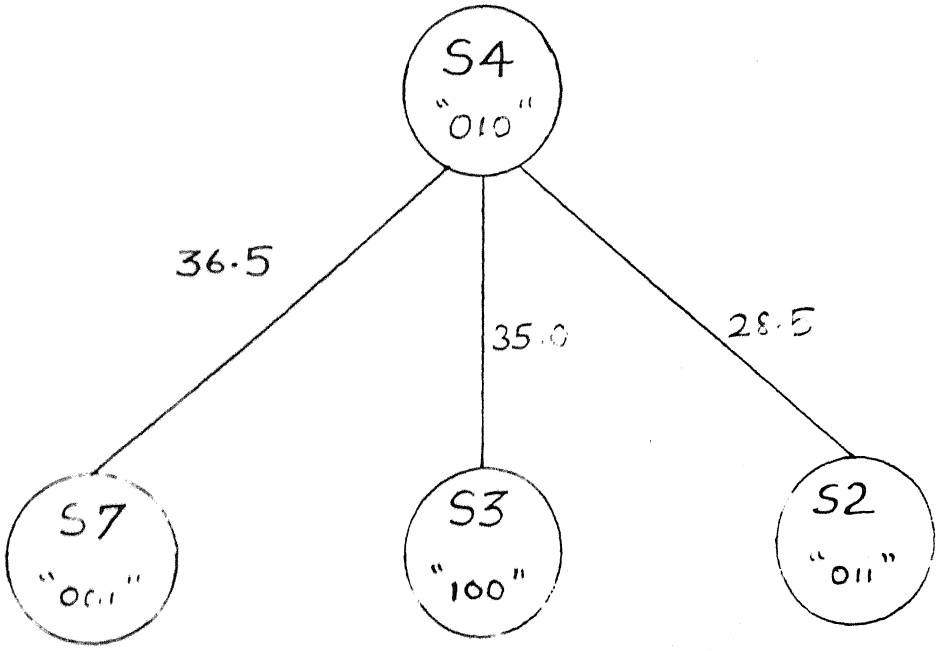


Fig.3.6 Clusters selected in the first two iterations of the graph embedding algorithm.



Now s6 is given the code "000" and s7,s4 and s3 unidistant codes from s6 ("001,010,100" respectively). Now s6 and all edges connected to s6 are deleted from the graph. Now, again sorting is done and a new graph evolves as is shown in Fig.3.4b. The cluster head is now s4 and the tail states are s7,s3 and s2. In this cluster, s4,s7 and s3 are already assigned codes. Hence s2 is given a unidistant code of "011" from s4. Then s4 and all edges to it are deleted from the graph and this process continues till all the nodes are given codes. The clusters are shown in Fig.3.6. The final assignment is shown in Fig.3.4c.

This procedure finally results in assigning codes which minimize the cost function as specified earlier.

CHAPTER 4

OPTIMAL ASSIGNMENT,TESTING AND PLA DECOMPOSITION

4.1 Introduction

The state assignment algorithms explained in the previous chapter have certain shortcomings in their approach. Even though only a STT in full conveys the necessary information, these algorithms split the STT for their operation. The fanin algorithm operates on the input and next state fields and completely ignores the output space, while the fanout algorithm operates on the present state and output fields while ignoring the input space. Hence the fanout algorithm works well for FSMs with a large number of outputs and small number of inputs. However, for a general machine, these two algorithms perform differently because they act on different fields of the same STT. Hence, a hybrid of the two algorithms is necessary, which will in effect operate on all the fields of the STT. In the embedding *wedge clustering* heuristic algorithm, after a cluster is formed, the initial code to the head of the cluster (if it is not already coded) is given randomly. In the first iteration, this means that the head node of the cluster can be assigned any code out of the total of 2^{N_b} codes, since for each code of N_b bits, there are N_b codes that are unidistant. However, assigning codes randomly to the head node affects the minimized cover of the optimized PLA. In this chapter, some optimality constraints for state assignment are presented. The variations from the algorithms explained in the previous chapter are discussed.

4.2 Hybrid Algorithms for State Assignment

The necessity for a hybrid algorithm is explained in the previous section. As has been explained in chapter 3, the fanout algorithm attempts to maximize the size of common cubes in the encoded machine prior to optimization. The fanin algorithm attempts to maximize the number of occurrences of the common cubes in the encoded machine prior to optimization. Also, cubes are the most common complex factors that can be found in a two-level network. Hence if the encoded machine prior to optimization has the most frequently occurring common cube as the largest common cube then this will result in a lot of good common factors in the encoded machine. The logic optimizer can extract these good common factors and give a minimal area/delay implementation of the PLA. This involves combining the weights, of the edges in the constructed complete graph, obtained by the fanin and fanout algorithms.

Since the fanin algorithm operates on the first and third fields of the transition table and the fanout algorithm on the second and fourth fields, the simplest way is to get a product of the two edge weights as the new edge weight for the complete graph. The fanin and fanout algorithms assign weights to the edges of the graph based on the proximity relations existing in the transition table. This means that, as has already been explained, the two states with a high edge weight when given close codes in the Boolean space result in providing a common factor in the encoded machine. Since the two algorithms operate

differently, it is possible that the proximity constraints between the two states (as indicated by the edge weights in the graph) may not be similar. In fact, experimental observations indicate that in some examples, the graphs are entirely different. Hence, obviously, the product of the two weights is a very good measure of the similarity between the two algorithms. Thus this approach produces a weighted graph which can then be given to the embedding heuristic for code assignment. However, in the course of the experimentation, it was also found that a weighted sum of the two edge-weights, as scaled by the number of inputs and number of outputs, performed reasonably in creating a good number of common cubes. These algorithms are collectively tested and the results indicated in chapter 5.

Symbolic minimization based approaches are effective in minimizing the cardinality of the minimal cover of the PLA. Since logic minimization of the FSM combinational component is applied before state assignment, a state assignment approach using this minimal representation should be much effective in creating common cubes. This is made use of in the algorithm.

The symbolic cover is a collection of symbolic implicants representing the state transitions. In general, a symbolic implicant can represent a transition from one or more states to a next state under some input conditions[8]. Therefore, there exist several symbolic cover representations that are equivalent among each other. A minimum symbolic cover is one of minimum cardinality. Symbolic minimization consists of finding such a

minimum symbolic cover, i.e., is equivalent to determining a minimum sum of products representation independent of the encoding of the symbolic strings.

A symbolic implicant such as

0 s1 s6 00

0 s4 s6 00

is represented in positional cube notation after symbolic minimization by 0 1001000 000001000. Thus, the effect of symbolic minimization is to group together the states that are mapped by some input into the same next state and assert the same output. The hybrid algorithm reads this grouping information for constructing the weighted graph.

The embedding algorithm has a definite randomness in the assignment of codes. In fact, the head of the cluster in the first iteration can be given any one of the all possible codes. This clearly is a drawback. For, the experimentation has shown that different codes to head nodes, or for that matter any other node in the graph, give different cardinalities of the cover of the PLA. To verify this fact and to obtain certain rules for optimal cover implementation, some examples from the MCNC FSM benchmark set were tested. Here the states were given all possible combinations of codes and the results were analyzed. Since for a n-state machine, there are n! factorial ways of giving codes, this experimentation was carried out only on the smallest machines, namely dk14,lion,mc,train4 .

The distance graphs(to be defined) are all shown to be isomorphic to each other for optimal cover.

A distance graph is a complete graph wherein the nodes correspond to the states of the machine and the edges have integers as labels which specify the Hamming distance between the codes of the two nodes connected to that particular edge.

The most striking observation found was that the distant graphs for optimum assignments are all isomorphic. This is a very important observation used to find the rules of encoding used in the hybrid algorithm to overcome to some extent, the randomness involved in the assignment of codes. The distant-graphs for the examples are shown in Figs.4.1 to 4.4.

It is seen from these graphs that certain states in the machine are to be coded with certain groups of codes only for achieving a optimal cover of the PLA. For example, in the example train4, s0 and s2 should be coded only from the set {00,11} for optimality. This type of all-zero coding eliminates some amount of randomness. The method by which the necessary all-zero code is given is explained below. If a particular next state produces an all zero output during a transition from some present state to that next state, then by giving that particular next state an all-zero code, the corresponding product term can be eliminated. However, in a STT description there may be many such next states . Then in that case, the next state with the maximum fanin and with maximum number of zeros in the output columns is

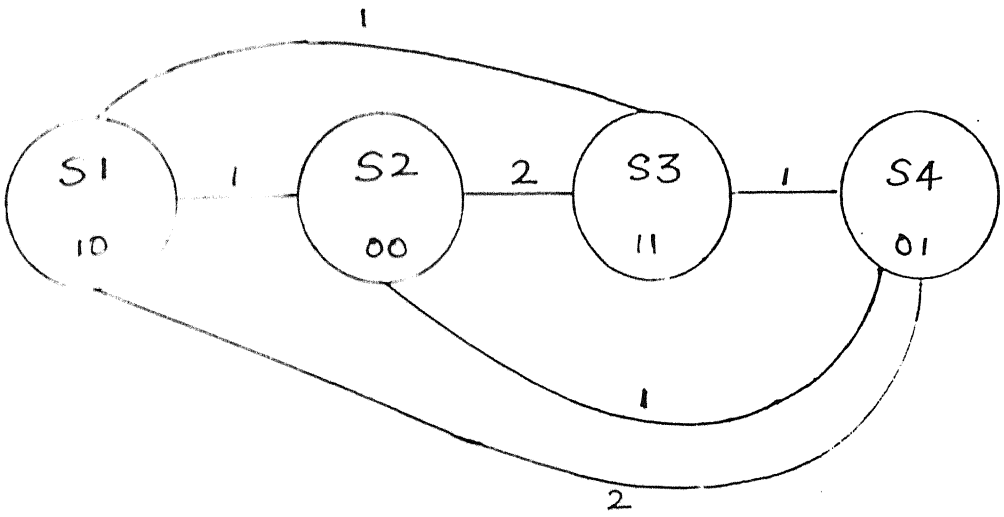
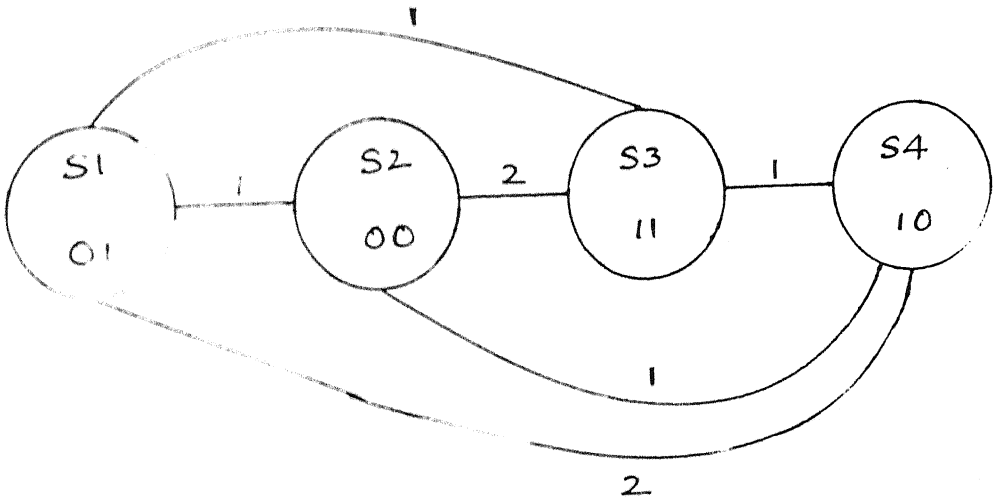


Fig.4.1 Distance graphs of dk15.



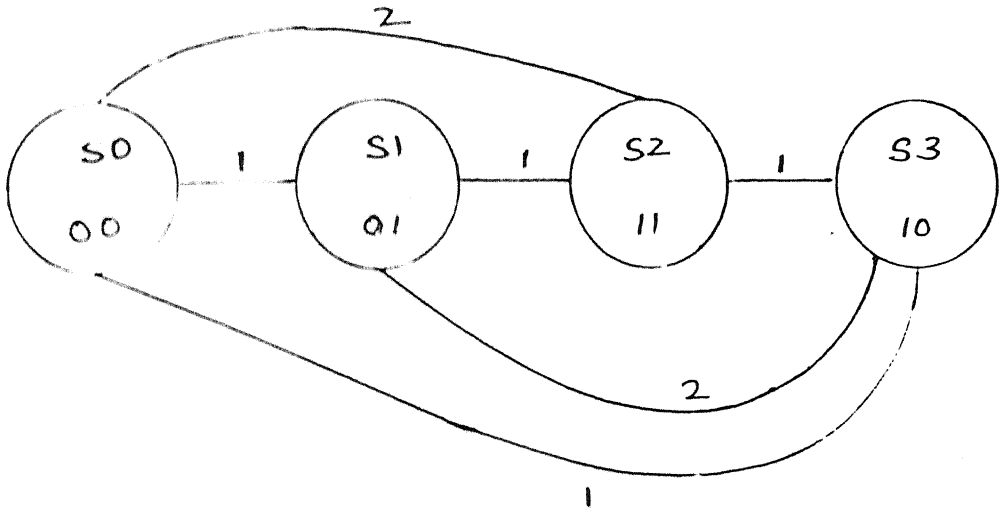
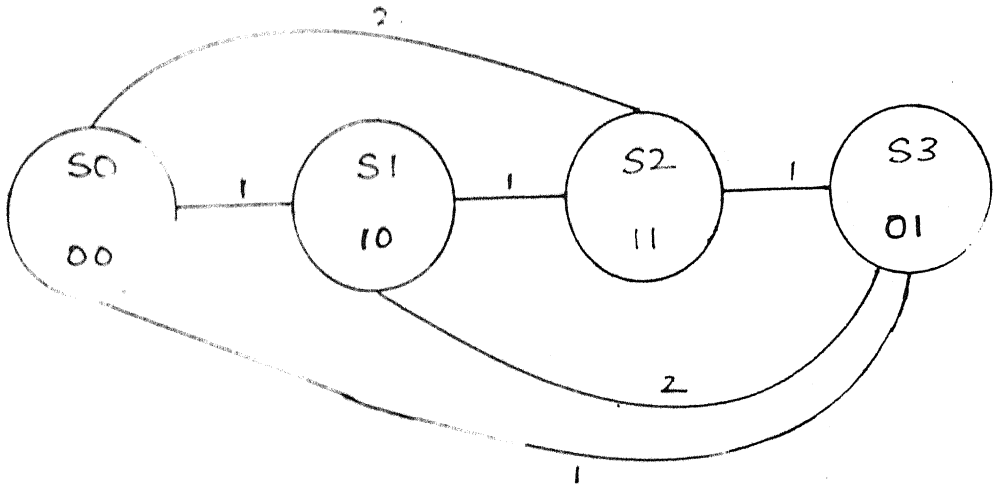


Fig.4.2 Distance graphs of lion.



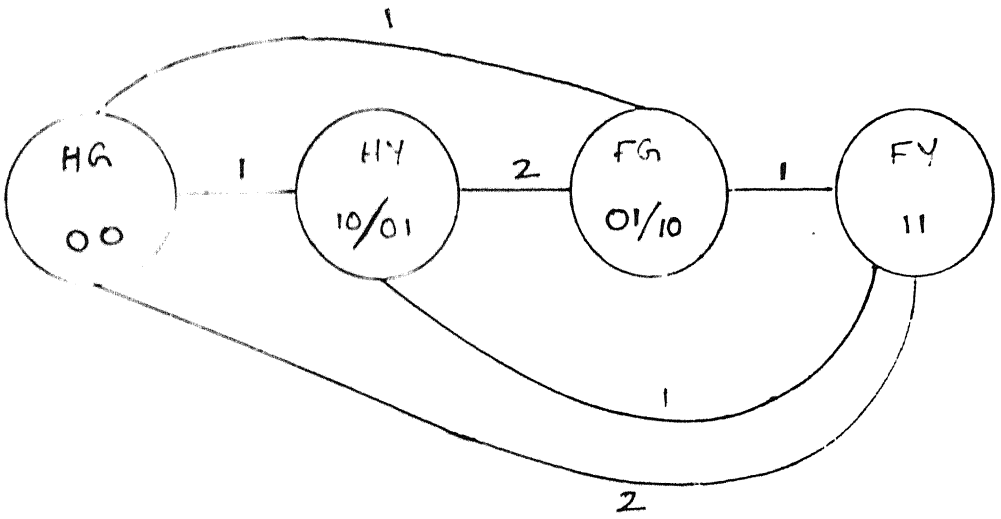
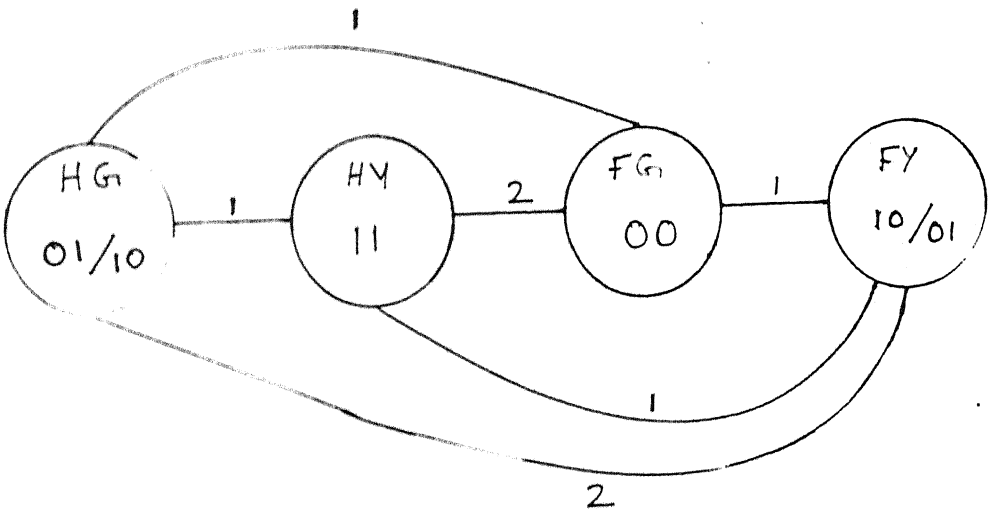


Fig.4.3 Distance graphs of mc.



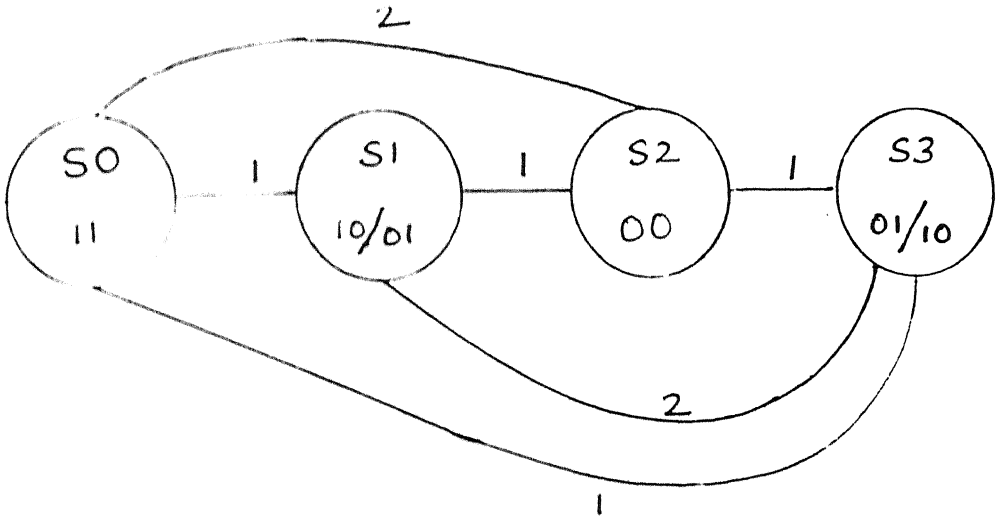
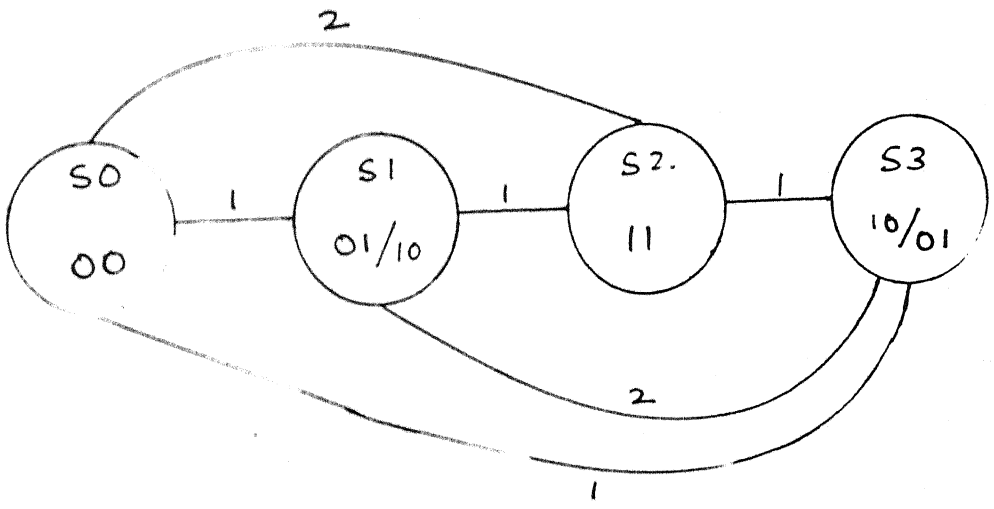


Fig.4.4 Distance graphs of train4.



chosen as the right candidate for all-zero coding. Then the graph is given to embedding heuristic for further assignment of codes. This all-zero coding does not affect the distance relations which are established by the algorithms for weight construction. Though these observations are based on some experimentation with small machines, the algorithms were tested on the MCNC FSM benchmark set and the results are tabulated in chapter 5.

4.3 Illustration

Consider the state transition table of the benchmark dk27. It is a single input, two output 7 state machine. The all-zero coding set is constructed as follows. For each next state, the number of times that particular transition from some present state to that next state producing all-zero outputs is calculated. In this example , the set is $\{s1^0, s2^0, s3^1, s4^1, s5^3, s6^2, s7^1\}$. Thus s5 is given the all-zero code and this will result in the definite elimination of atleast 3 product terms by the logic minimizer.

If the STT does not contain all-zero outputs transition at all, the state with maximum fanin and has maximum number of zero-column outputs is given the zero-code, as in general, the logic minimizer will find a minimum cover for such a sparse description.

4.4 Delay Calculation [18]

Optimization for performance is sometimes an overriding

```
# The name of this benchmark is dk27 .
.i 1
.o 2
.p 14
.s 7
0 s1 s6 00
0 s2 s5 00
0 s3 s5 00
0 s4 s6 00
0 s5 s1 10
0 s6 s1 01
0 s7 s5 00
1 s6 s2 01
1 s5 s2 10
1 s4 s6 10
1 s7 s6 10
1 s1 s4 00
1 s2 s3 00
1 s3 s7 00
.e
```

Fig.4.5 STT of MCNC FSM benchmark dk27.

criterion than minimum area implementation itself. Hence the calculation of delay in a PLA is very important. The delay in a PLA cannot be exactly determined without actually taking into account the physical layout. This is because, except for the cardinality of rows and optionally the number of devices, all other details like interconnection length are dependent on the implementation topology and the physical layout. Hence, for a given truth table description of a PLA, the most commonly used[18] delay measure in PLAs is the longest-delay path. To approximate the delay in a PLA, the following model is assumed ; the input to the AND plane of the PLA would be from two buffers, one providing the true form and the other complemented form. The delays in all the devices are assumed to be a constant.

The longest delay path is computed in the following way. The maximum delay in both the planes of the PLA is due to the maximum number of devices in that path. In the AND plane, the maximum delay line corresponds to the input line which has the maximum fanout. This line corresponds to the literal in the Boolean function of the PLA which is present in the maximum number of product terms. The maximum input fanout for literal is is thus given by

$$f_{I_i} = K_{I_i} P$$ where P is the number of product terms and the ratio factor K_{I_i} varies between 0 and 1. The maximum delay path in the OR plane of the PLA is through the output line from the AND plane which has the maximum fanout. The maximum fanout of the AND plane is given by $f_{A_j} = K_{A_j} O$ where O is

the number of outputs and the ratio is similarly defined. The total delay for the PLA is given by $D_{PLA} = max(f_{I_i} + f_{A_j})$ where the max is over all pairs (i,j) that form a path from the input to the output of the PLA.

It is seen that the delay is directly proportional to the number of product terms in the PLA. However, this is only a weak approximation, since the actual longest delay path depends on the number of devices in the path only and not on the number of product terms itself.

4.5 Testing of PLAs [19]

Testing of PLAs is an important step in the design cycle to ensure an error free and reliable circuit. Although automatic test pattern generation is in general very time consuming and difficult, it is , however possible to exploit the regularity of PLAs to devise algorithms which are more efficient than the ones available for random logic. The algorithms implemented in [19], are very efficient and generates very compact test set even for large PLAs. These are explained below.

4.5.1 Fault models for PLAs

Faults in PLAs are classified as stuck-at, bridging and cross-point faults.

stuck-at faults : Here, one of the lines is stuck permanently at one of the two logic values.

bridging faults : This fault is a short between two adjacent or crossing lines.

cross-point faults : This fault is due to the absence or the unnecessary presence of a cross connection or device between a bit line and a product line or between a product line and a sum line.

The effects of cross-point faults within the PLA can be represented functionally as changes in the product terms of the PLA and it may cause one of the following four effects.

INPUT PLANE :

G (Growth/missing contact faults)

S (Shrinkage/extra contact faults)

OUTPUT PLANE :

A (Appearance/extra contact faults)

D (Disappearance/missing contact faults)

A missing device in the AND plane causes a literal to disappear from a product term and hence it grows(Growth fault). An extra device in the AND plane causes a literal to appear in a product term and hence it shrinks(Shrinkage fault). An extra device in the OR plane causes a product term to appear in a function(Appearance fault). A missing device in the OR plane causes a product term to disappear from the function.

4.5.2 Fault Detection

Any minterm covered by either growth term or product term, but not both qualifies as a test vector for the growth fault. In order to propagate the fault to output through kth column of

OR-plane, the test vector must be free from all other product terms which contribute to the kth output. Shrinkage faults can be detected by method similar to Growth faults, since this fault causes the don't cares in the product term to become 0 or 1. The Appearance/Disappearance of a product term from an output column can be detected by a free minterm. A minterm covered by a product term is said to be free if it is not covered by any other product term of the function under consideration.

The basic principle behind the algorithms implemented in PLATEST[19] is path sensitization and deductive fault simulation. First a test is generated, using the path sensitization approach to detect a fault. Then the size of the test set is minimized using two heuristics, fault sorting and don't care fixing. Fault sorting basically is concerned about the order of fault processing. Using a thumb rule that a fault site closer to a primary output has more tests, the heuristic first considers faults with fewer tests to reduce the test set. Don't care bit fixing is nothing but specifying don't care bits as 0 or 1. This is done based on a merit function that sums up all the number of undetected faults on the bit column for all cubes. This increases the chances of accidentally detecting faults other than the one that it is designed for.

These algorithms, implemented in a program called PLATEST, are used to determine the test vector set for a PLA.

4.6 Decomposition of PLAs [20]

Area and speed optimization can be achieved by decomposing a single two-level Boolean function into two levels of cascaded PLAs. It has been shown that [21] the cardinality of Boolean function, whose inputs represent different values of multiple-valued variables, can be much smaller than the cardinality of equivalent function with purely binary inputs. This concept has been applied to the minimization of PLAs with two-bit decoders[21]. The same idea is applied to the minimization of two-level Boolean functions by means of PLA decomposition[20].

In [20], a single two-level PLA is decomposed into two levels of cascaded PLAs, such that the total area is smaller than that of the original PLA. The first level may contain an arbitrary number of PLAs : PLA11,PLA12,...PLA1m, and the second level contains a single PLA,PLA2 as shown in Fig.4.6. The first level PLAs can be viewed as generalized or programmable decoders, each with arbitrary number of inputs and outputs. The structure of the PLAs is not limited to that of standard decoders.

The prominent aspects of generalized decomposition are :

a) PLA is decomposed into a two-level set of PLAs. There is no fixed number of first-level PLAs and also no constraint on the number of inputs to any of the first-level PLAs.

b) Graph partitioning techniques are used for partitioning of input variables. This allows the minimization of overall PLA

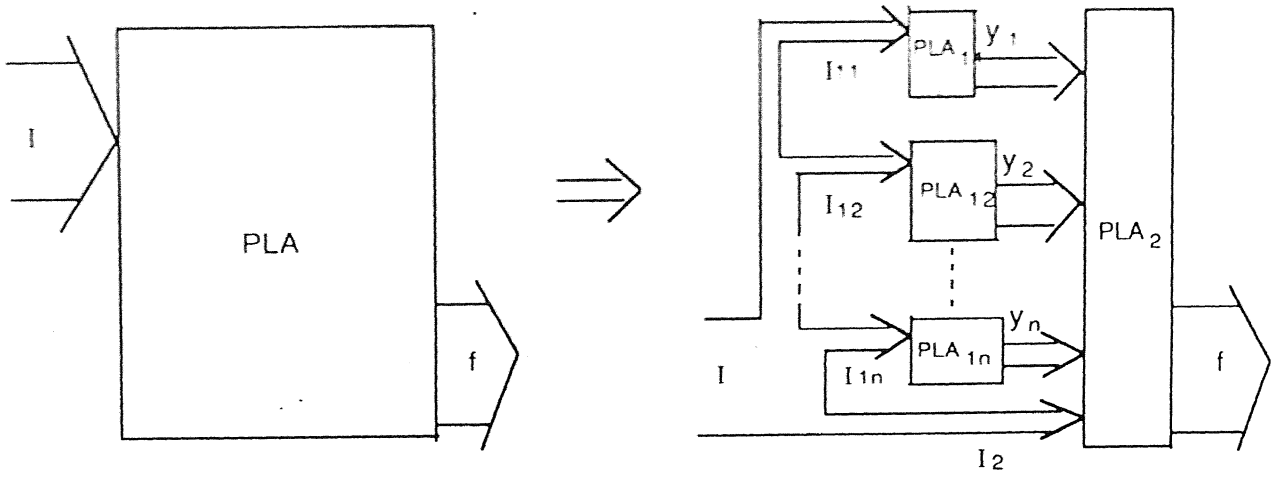


Fig.4.6 The structure of decomposed PLA.

area, including the area of first-level PLAs.

c) An efficient output encoding of first-level PLAs based on the concept of compatibility of dichotomies is used. A Graph covering algorithm is used to solve the encoding problem.

The PLA-based FSMs after state assignment is represented as a single PLA. The PLA decomposition tool is used to find out the effects of state assignment on the eventual decomposed area.

The tool takes a minimized multiple-valued PLA description. A modified form of the graph partitioning technique proposed by Kernighan and Lin in [22] is used to partition the input variables. The next step is the encoding process, wherein each combination of the binary inputs is encoded by a distinct binary pattern. Encoding of minimum length is made in order to minimize the size of the PLA. This process is repeated over all the sets of the input partition. Then a minimized cover of all the PLAs is chosen so that the sum of all the individual areas is minimum.

CHAPTER 5

RESULTS AND CONCLUSIONS

5.1 Logic Minimization

Logic minimization is an important step in the design of VLSI circuits. The encoded STT is usually given to logic optimizers for area/speed optimization. The most commonly used logic minimizer/optimizer is ESPRESSO[24] developed at Berkeley. The algorithms of ESPRESSO achieve a high level of efficiency mainly through the use of a "unate recursive paradigm". Based on this paradigm, a logic function is recursively divided until each part becomes unate. The sequence of operations carried out by ESPRESSO is outlined below.

1) Complement (Compute the complement of the PLA and the don't care set, i.e., compute the off-set) .

2) Expand (Expand each implicant into a prime and remove covered implicants).

3) Essential primes (Extract the essential primes and put them in the don't care set).

4) Irredundant cover (Find a minimal irredundant cover).

5) Reduce (Reduce each implicant to a minimum essential prime implicant).

6) Iterate 2,4 and 5 until no improvement.

7) Lastgasp (Try reduce, expand and irredundant cover one last time using a different strategy. If successful, continue the iteration).

8) Makesparse (Include the essential primes back into the cover and make the PLA structure as sparse as possible).

Although some of these procedures are similar to other logic minimizers, efficient Boolean manipulation is achieved using the "unate recursive paradigm", which is employed in complementation, tautology and other algorithms. Thus the performance of ESPRESSO as a logic minimizer is regarded as the benchmark. In this work, this logic minimizer has been extensively used to get the final minimized and irredundant cover of the PLA.

5.2 Results

The statistics of the MCNC FSM benchmark set is given in Table.5.1 . Table.5.2 gives all the statistics and results about DIET. As can be seen from this table, DIET uses, on most of the occassions, an encoding length which is greater than the minimum encoding length necessary. The columns *enclen* and *minlen* give this comparison. Table.5.3 shows the results of combining the weights of the fanin and fanout algorithms on the area count. Since, all the three algorithms use the same encoding length (which happens to be the necessary minimum), the product term count comparison is sufficient. This table shows the comparison of the algorithms. Table.5.4 and Table.5.5 show the comparison in terms of delay and testability respectively. Table.5.6 shows a comparison between the actual PLA areas of DIET and the hybrid algorithms. It is seen that the hybrid algorithm performs reasonably well in most of the cases. The improvement in area

Name	PIs	POs	States
bbara	4	2	10
bbsse	7	7	16
bbtas	2	2	6
dk15	3	5	4
dk27	1	2	7
dk512	1	3	15
ex2	2	2	19
ex7	2	2	10
fetch	9	15	26
keyb	7	2	19
lion	2	1	4
mc	3	5	4
modulo12	1	1	12
opus	5	6	11
planet	7	19	48
sand	11	9	32
sse	7	7	16
tbk	6	3	32
train11	2	1	11
train4	2	1	4

Table.5.1 Statistics of MCNC FSM benchmarks.

Name	inp	out	stats	minlen	enclen	pla_in	pla_out	prods	Area	Delay	Test
bbara	4	2	10	4	5	9	7	24	600	12	54
bbsse	7	7	16	4	6	13	13	28	1092	23	94
bbtas	2	2	6	3	3	5	5	13	195	10	24
dk15	3	5	4	2	4	7	9	17	391	15	39
dk27	1	2	7	3	4	5	6	9	144	8	19
dk512	1	3	15	4	5	6	8	18	360	11	33
ex2	2	2	19	5	6	8	8	29	696	19	66
ex7	2	2	10	4	6	8	8	15	360	14	52
fetch	9	15	26	5	5	14	20	32	1536	20	90
keyb	7	2	19	5	8	15	10	48	1920	46	238
lion	2	1	4	2	2	4	3	6	66	5	12
mc	3	5	4	2	2	5	7	9	153	8	17
modulo12	1	1	12	4	4	5	5	15	225	9	23
opus	5	6	10	4	4	9	10	16	448	16	-
planet	-	-	-	-	-	-	-	-	-	-	-
sand	11	9	32	5	6	17	15	101	4949	53	387
sse	7	7	16	4	6	13	13	28	1092	23	94
tbk	-	-	-	-	-	-	-	-	-	-	-
train11	2	1	11	4	5	7	6	10	200	10	39
train4	2	1	4	2	2	4	3	6	66	6	14

Table.5.2 Statistics and results of DIET.

Name	Fanin	Fanout	Hybrid	% Improvement	
				Hybrid vs Fanin	Hybrid vs Fanout
bbara	26	28	26	0	7.1
bbsse	34	33	33	-3.0	0
bbtas	14	10	9	35.7	10.0
dk15	21	18	21	0	-16.7
dk27	10	8	8	20.0	0
dk512	23	21	21	8.7	0
ex2	36	43	35	2.8	18.6
ex7	16	19	17	-6.2	10.5
fetch	34	34	33	2.9	2.9
keyb	58	107	55	5.1	48.6
lion	7	8	7	0	12.5
mc	9	10	10	-11.1	0
modulo12	14	14	14	0	0
opus	16	16	15	6.2	6.2
planet	97	99	98	-1.0	1.0
sand	104	110	99	4.8	10.0
sse	32	37	33	3.1	10.8
tbk	179	238	164	8.4	31.1
train11	13	16	14	-7.7	12.5
train4	6	6	6	0	0

Table.5.3 Comparison of algorithms : area .

Name	Fanin	Fanout	Hybrid
bbara	12	18	12
bbsse	29	27	27
bbtas	9	6	6
dk15	16	13	16
dk27	10	6	6
dk512	15	13	13
ex2	23	26	23
ex7	13	12	12
fetch	22	21	19
keyb	50	56	49
lion	6	6	6
mc	9	8	8
modulo12	10	10	10
opus	20	20	20
planet	56	69	62
sand	77	73	67
sse	27	25	29
tbk	120	133	122
train11	11	13	11
train4	5	5	5

Table.5.4 Comparison of algorithms : delay

Name	Fanin	Fanout	Hybrid
bbara	61	63	61
bbsse	117	126	115
bbtas	23	20	20
dk15	30	28	30
dk27	14	15	14
dk512	32	31	30
ex2	87	86	75
ex7	40	40	40
fetch	108	115	86
keyb	221	328	231
lion	13	14	13
mc	21	22	22
modulo12	22	22	22
opus	64	59	55
planet	296	295	273
sand	389	402	365
sse	111	123	110
tbk	563	581	572
train11	40	46	46
train4	14	14	14

Table.5.5 Comparison of algorithms : test

Name	Hybrid	DIET	Improvement
bbara	572	600	4.7
bbsse	1089	1092	0.3
bbtas	135	195	30.7
dk15	306	391	21.7
dk27	104	144	27.7
dk512	357	360	0.8
ex2	735	696	-5.6
ex7	288	360	20.0
fetch	1584	1536	-3.1
keyb	1705	1920	11.2
lion	77	66	-16.6
mc	153	153	0
modulo12	210	225	6.6
opus	420	448	6.2
planet	4947	-	-
sand	4554	4949	7.9
scf	1056	1092	3.3
tbk	4920	-	-
train11	221	200	-10.5
train4	66	66	0

Table 5.6 Comparison of hybrid algorithm with DIET

measure ranges from 0.3% to 30.7%. Only in 4 cases were the results in negative. It is to be noted here that the area measure used is the product of the total number of columns and the total number of rows of the PLA.

Table.5.7,5.8 and Table.5.9 all show the area, delay and test vector size comparison amongst the algorithms. Here the all-zerocoding strategy is adopted. Table5.10 shows a comparison between all-zerocoding hybrid algorithm and DIET. Here also, the improvements in area measure over DIET range from 0.8% to 27%. And only 3 examples failed to give improvement in area measure.

The decomposition of PLAs is performed and the results tabulated in Table.5.11. Here again the hybrid algorithm performs well. The area is calculated in the decomposed PLAs by means of constructing a binary format description from a multi-valued positional cube notation[25]. This involved some modifications in the output-data representation of the tool DEPLA. Table.5.12 shows the comparison with DIET. The results are extremely good. The improvements in area measure range from 3.3% to 38.5%.

5.3 Conclusion and Scope for future work

The state assignment algorithms are implemented in a module called *hyb_asgn*. This module has the option of selecting the fanin, fanout, hybrid(non all-zerocoding and all-zerocoding versions) algorithms. Since the input format of *hyb_asgn* is of the STT type, the module *hyb_rec* converts the multi-valued

Name	Fanin	Fanout	Hybrid
bbara	28	28	25
bbsse	34	35	31
bbtas	12	13	11
dk15	21	18	21
dk27	9	10	8
dk512	21	20	21
ex2	43	44	43
ex7	18	17	17
fetch	34	32	33
keyb	58	58	55
lion	7	7	7
mc	9	9	9
modulo12	15	15	15
opus	16	16	15
planet	96	99	96
sand	106	106	102
sse	34	34	31
tbk	161	196	164
train11	14	13	10
train4	6	7	6

Table 5.7 Comparison of zerocoding algorithms : area

Name	Fanin	Fanout	Hybrid
bbara	16	18	12
bbsse	29	30	27
bbtas	8	10	8
dk15	16	13	16
dk27	8	8	6
dk512	15	11	13
ex2	24	22	23
ex7	15	14	12
fetch	18	21	18
keyb	50	50	49
lion	6	6	6
mc	8	8	8
modulo12	9	9	9
opus	20	20	20
planet	57	66	60
sand	75	74	69
sse	29	30	27
tbk	115	130	120
train11	12	11	9
train4	6	5	6

Table.5.8 Comparison of zerocoding algorithms : delay

Name	Fanin	Fanout	Hybrid
bbara	65	61	53
bbsse	130	136	104
bbtas	20	20	21
dk15	30	28	30
dk27	15	15	14
dk512	31	30	30
ex2	91	84	85
ex7	45	43	41
fetch	104	108	97
keyb	221	272	231
lion	13	13	13
mc	20	20	20
modulo12	24	24	24
opus	68	57	53
planet	285	295	270
sand	406	388	373
sse	130	117	104
tbk	552	627	566
train11	42	43	33
train4	16	14	16

Table.5.9 Comparison of zerocoding algorithms : test

Name	Hybrid	DIET	Improvement
bbara	550	600	8.3
bbsse	1023	1092	6.3
bbtas	165	195	15.3
dk15	357	391	8.7
dk27	104	144	27.7
dk512	357	360	00.8
ex2	903	696	-29.7
ex7	306	360	15.0
fetch	1584	1536	-3.1
keyb	1705	1920	11.2
lion	77	66	-16.7
mc	153	153	0
modulo12	225	225	0
opus	420	448	6.2
planet	4896	-	-
sand	4692	4949	5.2
sse	1023	1092	6.3
tbk	4920	-	-
train11	170	200	15.0
train4	66	66	0

Table.5.10 Comparison of zerocoding hybrid algorithm with DIET.

Name	Fanin	Fanout	Hybrid
bbara	405	405	385
bbsse	1203	1089	891
bbtas	150	135	120
dk15	272	272	272
dk27	117	91	91
dk512	357	340	306
ex2	735	882	714
ex7	288	324	270
fetch	1584	1584	1536
keyb	1254	2387	1305
lion	55	88	55
mc	148	148	148
modulo12	160	172	184
opus	448	420	420
planet	4896	4743	4692
sand	4416	4738	4416
sse	1203	1089	891
tbk	4096	4590	3750
train11	238	272	204
train4	55	55	55

Table.5.11 Comparison of algorithms with respect to area of the decomposed PLA.

Name	Decomp	DIET	Improvement
bbara	385	600	35.8
bbsse	891	1092	18.4
bbtas	120	195	38.5
dk15	272	391	30.4
dk27	91	144	36.8
dk512	306	360	15.0
ex2	714	696	-2.5
ex7	270	360	25.0
fetch	1536	1536	0
keyb	1305	1920	32.0
lion	55	66	16.7
mc	148	153	3.3
modulo12	184	225	18.2
opus	420	448	6.3
planet	4692	-	-
sand	4416	4949	10.8
sse	891	1092	18.4
tbk	3750	-	-
train11	204	200	-0.5
train4	55	66	16.7

Table.5.12 Comparison of hybrid algorithm with DIET with respect to the area of the decomposed PLA.

symbolically minimized transition table into a STT description.The delay calculations and decomposed area calculations are performed by the modules *hyb_del* and *hyb_decomp* respectively.

The state assignment algorithms were tested on the MCNC FSM benchmark set examples. The algorithms perform reasonably well and simultaneously reduces the longest-path delay measure and the size of the test set needed to test the PLA. These algorithms are optimal for area and can be used for efficiently designing PLA-based FSMs.

Future work on the same lines could include a totally new embedding strategy for a deterministic encoding procedure. A very efficient method of constructing the weighted graph, which tries to accurately model the main operations of the two-level logic optimizer. Also, work on performance-oriented decomposition and state assignment for decomposition can be taken.

REFERENCES

1) S.Devadas,"A synthesis and optimization procedure for fully and easily testable sequential machines", IEEE T-CAD, OCT '89.

2) W.Wolf, et al.," A kernel-finding state assignment algorithm for multi-level logic ", Proc. of 25th DAC.

3) J.Hartmanis and R.J.Stearns, Algebraic structure theory of sequential machines, Prentice-Hall,1966.

4) R.Karp, "Some techniques for state assignment for synchronous sequential machines" , IEEE T-COMP , OCT '64.

5) Z.Kohavi , "Secondary state assignment for sequential machines ", IEEE T-COMP, JUN '64.

6) D.B.Armstrong , "On efficient assignment of internal codes to sequential machines ",IEEE T-COMP OCT'62.

7) T.A.Dolotta and E.J.McCluskey, "The coding of internal states of sequential machines" ,IEEE T-COMP, OCT '64.

8) G.D.Micheli, et al., "Optimal state assignment for finite state machines" , IEEE T-CAD JULY ;85.

9) G.Saucier, "State assignment of asynchronous sequential machines using graph techniques" , IEEE T-COMP, MAR '72.

10) T.Villa , et al., " NOVA : State assignment of FSM for optimal two-level logic implementation",IEEE T-CAD,SEP 90.

11) S.Yang and M.Ciesielski, "Optimum and sub-optimum algorithms for input encoding and its relationship to logic minimization, IEEE T-CAD JAN '91.

12) S.Devadas et al., "Easily testable PLA-based FSMs" , IEEE T-CAD JUN '90.

13) K.Barlett et al., "Multi-level logic minimization using implicit don't cares",IEEE T-CAD JUN '88.

14) S.Devadas et al., "Irredundant sequential machines via optimal logic synthesis" , IEEE T-CAD JAN '90.

15) B.Prakash, "Easily Testable PLA-based FSMs ", M.Tech thesis, Dept. of EE, IIT-Kanpur, FEB '92.

16) R.Puri, "PLASMA : A CAD tool for state machine synthesis",M.Tech Thesis, Dept. of EE, IIT-Kanpur,APR '90.

17) S.Devadas et al., " MUSTANG : State assignment of FSM targeting multi-level logic",IEEE T-CAD, DEC '88.

18) Z.Hasan and M.Ciesielski, " FSM decomposition for performance optimization", to be published.

19) T.S.Raghuram, " Fault modelling and testing of PLAs", M.Tech thesis, Dept. of EE, IIT-Kanpur, FEB '89.

20) S.Yang and M.Ciesielski, " PLA decomposition with generalized decoders", TR-89-CSE-8, U MASS, Amherst, 1989.

21) T.Sasao , " Input variable assignment and output phase optimization of PLAs",IEEE T-COMP, OCT '84.

22) B.Kernighan and S.Lin, "An efficient heuristic procedure for partitioning graphs",BSTJ , FEB '70.

23) G.Rajagopalan, "Design of VLSI modules using Decoded PLA", M.Tech thesis, Dept. of EE, IIT-Kanpur, APR '89.

24) R.Brayton et al., Logic minimization algorithms for VLSI synthesis, Kluwer Academic, 1986.

25) DEPLA DOCUMENTATION, University of Massachussetts, Amherst, 1989. (The Nelsis IC Design System, IIT-Kanpur).

EE-1992-M-RAM-OPT